FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Brief Survey of Visual Programming Languages

Maria João Pombinho Miranda

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Hugo José Sereno Ferreira

23 de Julho de 2018

A Brief Survey of Visual Programming Languages

Maria João Pombinho Miranda

Mestrado Integrado em Engenharia Informática e Computação

23 de Julho de 2018

Resumo

O ser humano pensa visualmente. Desde os tempos mais remotos da história da humanidade que o ser humano usa imagens como método de comunicação entre si. Como tal, parece evidente que, num mundo já tão explorado como o da programação, a humanidade tenha criado formas visuais de se expressar. Quando programadores explicam a alguém a utilidade/usabilidade de um determinado programa, muitas vezes acabam por usar uma representação gráfica de um fluxo de controle, com caixas e setas. Sendo este tipo de representação de fácil compreensão, por que não usá-lo para o desenvolvimento de programas?

É neste sentido que surge o conceito de *Visual Programming* (VP) e de *Visual Programming Languages* (VPLs), que têm como objetivo tornar a programação mais acessível e reduzir as dificuldades que os *novice programmers* encontram permitindo que apenas se foquem na lógica do seu programa. Como tal, tanto para os *expert programmers* como para os *end-users*, as vantagens deste tipo de mecanismo tornam-se evidentes. No entanto, há muitas outras características destas linguagens que podem tornar-se como limitações para os programadores quando tentam elaborar os seus programas.

Assim, o objetivo desta dissertação passa por explorar a área da *Visual Programming* a partir da perspetiva do *end-user*. Perceber quais as vantagens e desvantagens da sua utilização e onde e quando a sua utilização poderá ser uma boa abordagem para os problemas com que os utilizadores se podem deparar. Serão então explorados diferentes domínios e diferentes perfis de utilizadores, e como cada um deles se relaciona com os diferentes tipos de VPLs.

Esta dissertação compreendeu 4 fases de desenvolvimento. Na fase inicial foram estudados todos os conceitos inerentes ao tema. Numa fase posterior foi feito um estudo às várias linguagens de programação visual. Depois disto foi elaborado um inquérito a vários utilizadores deste tipo de linguagens. Finalmente, foram analisados todos os resultados obtidos nos inquéritos e foram retiradas as devidas conclusões.

Agradecimentos

A referência de sempre ao apoio dos meus pais e irmãos.

Ao meu amigo José Cardoso por toda a disponibilidade, ajuda e sobretudo pela amizade.

Aos professores Luís Filipe Teixeira e João Pascoal Faria por todo o apoio e compreensão.

Ao meu orientador, Professor Hugo Sereno Ferreira, por toda a preocupação, ajuda e orientação.

Maria João Pombinho Miranda

"If I have seen further it is by standing on the shoulders of giants"

Isaac Newton

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

VP	Visual Programming
VPL	Visual Programming Languages

Capítulo 1

Introdução

Este capítulo introduz o relatório técnico desta dissertação. Inicialmente é descrita a contextualização do problema em questão seguindo-se da sua identificação. Depois disto é abordada a motivação associada e os objectivos. Por último, será feita uma descrição da estrutra do presente relatório.

1.1 Contextualização

Desde os tempos mais remotos da história da humanidade que o ser humano usa imagens como método de comunicação entre si. Como tal, quando ponderamos sobre como está no momento o "Mundo da Programação" relativamente ao usa de linguagens de programação para desenvolvimento de Software uma série de questões se levantam. Por que é que se continua a tentar comunicar com os nossos computadores usando linguagens de programação textuais? O poder dos computadores modernos não seria mais acessível a um número maior de pessoas se pudéssemos instruir um computador apenas com as imagens que desenhamos na nossa mente quando pensamos nas soluções para os problemas? Proponentes das *Visual Programming Languages* argumentam que a resposta a ambas as perguntas é sim. Inicialmente as pessoas pensam nas coisas em termos de imagens, usam a imagem como um componente primário do pensamento criativo. Nas secções seguintes são abordados os conceitos de *Visual Programming*, *Visual Programming Languages*, bem como exemplos e aplicações de ambos os conceitos.

1.2 Objetivos e Motivação

O objetivo desta dissertação passa por explorar a área da *Visual Programming* a partir da perspetiva do *end-user*, focando-me nas metáforas utilizadas na interface gráfica do utilizador e na *User Experience* associada. Para além disto, serão também explorados diferentes domínios e diferentes utilizadores, utilizadores com graus diferentes de conhecimento quanto à programação.

Com isto pretende-se perceber quais as linguagens que se aplicam quando temos um determinado domínio e um determinado tipo de utilizador. Pretende-se obter um estudo sobre a utilização deste tipo de linguagens nos dias de hoje e a sua aceitação no mundo dos programadores. A motivação com este estudo passa por perceber a aceitação das *Visual Programming Languages* nos diferentes domínios e para diferentes utilizadores. Pretende-se também perceber como funcionam algumas das linguagens.

1.3 Estrutura da Dissertação

Este relatório encontra-se organizado em 5 capítulos, sendo que o primeiro é focado na introdução e contextualização do problema que levou à necessidade da elaboração desta dissertação bem como a motivação e objectivos propostos. No segundo capítulo são descritos todos os conceitos inerentes ao tema a tratar. No terceiro capítulo é apresentado o Estado da Arte relativo às Linguagens de Programação Visual estudadas e algumas aplicações das mesmas. No quarto capítulo são analisados os resultados de inquéritos que elaborei para o meu estudo com recurso a uma determinada amostra que será também descrita no mesmo. Por fim, no último capítulo, é descrita a metodologia usada para resolução do problema, bem como o trabalho futuro e respectivas conclusões.

Capítulo 2

Estado da Arte

Ao longo deste capítulo serão mostradas algumas definições de Programação Visual, será descrita a diferença entre Linguagens de Programação Visual e Ambientes de Programação Visual. Para além disto, será feita uma caracterização deste tipo de linguagens e será apresentada a sua classificação. Por último, será feita uma abordagem aos prós e contras da utilização deste tipo de linguagens.

2.1 Visual Programming - Definição

Myers definiu *Visual Programming* (VP) como qualquer sistema que permita aos utilizadores especificar um programa de duas ou mais dimensões. As linguagens de programação convencionais não são consideradas bidimensionais, uma vez que o compilador ou interpretador as processa como sendo um fluxo longo e unidimensional. A *Visual Programming* não inclui sistemas que usem linguagens de programação convencionais para definir imagens. [Mye86]

Conforme descrito em [BBB+94] o objetivo deste tipo de programação passa por facilitar o processo de expressar e compreender os programas, tanto para um utilizador que esteja habituado a trabalhar com este ou outro tipo de programação como para *novice programmers*, que não têm quaisquer bases de programação. Isto é conseguido pois com este tipo de linguagens de programação o número de conceitos que fazem parte da linguagem é reduzido, permitindo assim que objetos de dados sejam explorados diretamente representando conexões e relacionamentos entre si. O facto de estas linguagens apresentarem um *feedback* visual imediato de todas as interações, cálculos e semântica também facilita bastante o trabalho dos programadores.

De acordo com Burnett [Bur99], Visual Programming é a programação onde mais do que uma dimensão é usada para transmitir semântica. Como exemplos destas dimensões adicionais são o uso de objectos multidimensionais, o uso de relações espaciais, ou até mesmo o uso da dimensão "tempo" para especificar as relações semânticas "antes e depois". Para Burnett, cada objecto ou cada relacionamento multidimensional significativo pode ser considerado como um token, assim

como nas linguagens de programação textual tradicionais, onde cada palavra é um token, sendo que o conjunto de um ou mais tokens é uma expressão visual. A multidimensionalidade é então para Burnett a diferença essencial entre Visual Programming Langiages e linguagens extritamente textuais.

Para Golin e Reiss [GR90] o termo linguagem visual é usado para descrever vários tipos de linguagens: linguagens que manipulam informações visuais; linguagens para apoiar interações visuais e linguagens para programação com expressões visuais, sendo esta última categoria a mais usada para referir linguagens de programação visuais. Para estes autores, as linguagens de programação visual e as linguagens de programação textual são semelhantes, uma vez que os programas são formados sobre um alfabeto básico e podem ser decompostos numa estrutura bem definida. As principais diferenças entre estes dois tipos de linguagens passa por as cadeias de texto serem unidimensionais e portanto possuem uma ordem linear, já uma imagem é um objecto bidimensional. Uma ordenação linear das componentes de uma imagem não preservaria a relação adjacente entre os elementos. A outra grande diferença importante para os autores entre imagens e texto é a estrutura subjacente. Explicam que a natureza essencialmente linear do texto é reflectida numa estrutura de árvore, já numa imagem, um subconjunto pode compor com várias outras formas, em vez de apenas um vizinho imediato à esquerda ou à direita. Assim, numa linguagem visual, a estrutura subjacente é um gráfico direccionado em vez de uma árvore.

Shu define as Linguagens de Programação visual como linguagens que usam algumas representações visuais, além de ou no lugar de palavras e números, para realizar o que teria de ser escrito numa linguagem de programação tradicional unidimensional. Para Shu, para uma linguagem ser considerada linguagem de programação visual, a própria linguagem deve usar algumas expressões visuais significativas como meio de programação.

Para Narayanan [NH97] Linguagens visuais tratam-se de idiomas com alfabetos constituídos por representações visuais e utilizados para comunicação entre humanos ou entre o ser humana e um computador. Programação visual trata-se do uso de representações visuais para comunicar a um computador dados e operações. E entende que uma linguagem de programação visual se trata de uma linguagem de programação com um alfabeto composto de representações visuais.

2.1.1 Estratégias das Visual Programming Languages

As *Visual Programming Languages* têm vindo a ser desenvolvidas para dar resposta às necessidades que o conceito de *Visual Programming* requer. Pode-se referir que as *Visual Programming Languages* tentam então combinar o poder de comunicação de uma linguagem visual com as possibilidades de uma linguagem de programação.

[Bur99] Sendo um dos principais objectivos dos designers das *Visual Programming Languages* melhorar a capacidade do programador de expressar a lógica dos seus programas e entender como estes funcionam, pois as VPLs têm menos restrições sintácticas proporcionando maior liberdade para a exploração de mecanismos de programas, estas linguagens têm então um conjunto de técnicas/estratégias comuns que criam um conjunto de carcterísticas importantes neste tipo de linguagens:

- ***Concretness* (Concretude):**. Em muitos VPLs, o programador pode ver, explorar e alterar valores de dados específicos ou até mesmo execuções, ou seja, usando objetos particulares a linguagem de programação visual permite ao programador especificar aspectos importantes de qualquer parte específica do programa.

- ***Directness* (Imediatismo):**. Estas linguagens permitem a manipulação de partes do programa diretamente pelo programador, reduzindo as ações necessárias para alcançar essa mudança quando comparado com o processo equivalente em linguagens textuais.

- ***Explicitness* (Clareza):**. Em linguagens textuais a relação entre objectos está implícita, em VPLs quanto mais explícitas são essas relações melhor. Um utilizador de uma determinada VPL deve ter a informação o mais explícita possível. Tudo o que há para ser conhecido sobre o programa pode ser facilmente entendido olhando para a representação gráfica e o utilizador não precisa de deduzir a semântica por si mesmo

- ***Immediate Visual Feedback* (Feedback Visual Imediato):**. Alterações no programa devem ser vistas e sentidas imediatamente, ajudando assim o programador a encontrar os seus erros mais facilmente e mais cedo.

2.1.1.1 Classificação das Visual Programming Languages

[BD04] Ao longo do tempo, a curiosidade sobre as VPLs foi aumentando e, como tal, houve um acréscimo nos estudos e pesquisas deste tema. Para ajudar os investigadores a encontrar trabalho na área, bem como para comparar e avaliar diferentes sistemas, surgiu a necessidade de classificar estas linguagens de uma forma robusta e padronizada. Como mostra o relatorio de Boshernitsan, estas linguagens podem ser classificadas em:

- ***Purely visual languages*:** Estas linguagens são caracterizadas pela dependência de técnicas visuais ao longo do processo de programação. O utilizador manipula ícones ou outras representações gráficas para criar um programa que posteriormente é depurado e executado no mesmo ambiente visual. O programa é compilado diretamente a partir da sua representação visual e nunca é traduzido numa linguagem provisória baseada em texto.

- ***Hybrid text and visual systems*:** Tentam combinar elementos visuais e textuais, incluindo tanto aqueles em que os programas sao criados visualmente, e, em seguida, traduzidos para uma linguagem de texto, como sistemas de texto de alto nivel que envolvem o uso de elementos graficos numa linguagem textual. Pode entao com este tipo de sistemas alternar entre os modos de visualizaçao.

- ***Example-based systems*:** [Mye90] De acordo com Brad A. e Myers, *Example-based programming* tem duas formas: *Programming by example* e *programming with example*, portanto o termo *example-based programming* pode ser mais geral do que o termo usado por Boshernitsan. Estes sistemas são baseados em exemplos de entrada e saída. *Programming by example*, ou programação automática, tenta inferir e completar o programa a partir de

exemplos de entrada ou saída ou excertos de execução. *Programming with examples*, exige que o programador especifique tudo sobre o programa, ou seja, desenvolver o programa na sua totalidade, a partir de um exemplo em específico.

- ***Constraint-oriented systems*:** Neste tipo de sistemas existe um conjunto de restrições construídas para criar regras num determinado ambiente, o que e utilizado para criar simulações, documentos dinâmicos e objetos gráficos manipuláveis.

- ***Form-based systems*:** Este sistema é caracterizado pela presença de células que possuem algum tipo de conexão entre elas, sendo que o exemplo mais conhecido deste tipo são as *spreadsheets*

2.1.2 Prós e Contras das Visual Programming Languages

Estando os computadores e a programação cada vez mais presente no dia-a-dia das pessoas e de uma forma cada vez mais evoluída, a utilidade das Visual Programming Language assim como as suas vantagens em comparação com as linguagens textuais, são um tema de grande discussão.

São vários os autores que apontam as vantagens e desvantagens, que a seu ver, a utilização destas linguagens trazem.

Como é referido no artigo [Ahm99], a Programação Visual permite que os utilizadores dominem a complexidade da programação visualizando-a. A passagem da abstração para o nível visual revela relações semânticas entre as entidades do programa tornando-o mais compreensível. O aumento da produtividade pode também ser considerado como um dos seus benefícios. Este aumento é obtido devido à maior facilidade da utilização deste tipo de sistemas ou também do aumento da comunicação entre os programadores e os utilizadores dos sistemas, uma vez que os utilizadores conseguem geralmente entender o processo inicial de projectar os sistemas.

O estudo de Green [GP92] comparou a legibilidade da programação textual e gráfica. Os resultados do mesmo foi que os programas gráficos demoravam mais a compreender do que os textuais. Foram usados utilizadores experientes tanto em programação visual como em programação por texto para este estudo, ou seja, o estudo não se concentrou na experiência do utilizador, como "End-user", mas sim na experiencia dos programadores. A maior parte das vantagens encontradas neste tipo de linguagens é nesta área, na experiência do utilizador.

Para Meyer, [Mye86] a programação visual é muito importante para efeitos pedagógicos, pois este tipo de linguagens pode fazer da programação uma actividade mais fácil e menos sujeita a erros, para não programadores conseguirem criar os seus próprios programas, o que através de linguagens textuais seria mais complicado, pois é sabido que as pessoas lidam muito melhor com exemplos do que com abstrações.

Capítulo 3

Descrição de Linguagens de Programação Visual

Neste capítulo serão apresentados três domínios, educacional, jogos e multimédia, aos quais estão associadas linguagens de programação visual. Serão então descritas essas linguagens e as suas principais características.

3.1 Domínio Educacional

O ensino de programação a utilizadores sem bases de programação é uma tarefa difícil, pois o desenvolvimento de um alto nível de abstração e raciciocínio lógico por parte destes utilizadores é um processo lento e gradual [Dij89]. A grande quantidade de informação e conceitos necessários a interiorizar nos primeiros tempos de aprendizagem, bem como a necessidade de aprender o raciocínio lógico, a codificação e a depuração, são barreiras que levam muitos destes utilizadores ao fracasso e à sua desistência [CPR04]. Para além disto, devido a complexidade de algumas linguagens de texto, estas podem nao ser as ferramentas mais adequadas para ensinar os *novice programmers*. As linguagens de programação visual do domínio educacional surgem assim com o objectivo de tornar a aprendizagem de programação num processo mais apelativo e fácil para os utilizadores. Várias pesquisas mostram que o uso de gráficos, por exemplo, em vez de texto, tornam o desenvolvimento de programas mais intuitivo. [Rod02] [Guz03] Nas subsecções seguintes serão apresentados três exemplos de *Visual programming languages* que se podem enquadrar neste domínio.

3.1.1 Scratch

[GLK] [Mal10] O Scratch é uma linguagem de programação visual que permite aos utilizadores criar projetos interativos, desde histórias animadas, jogos, tutoriais, projetos de música, etc. A

gramática do Scratch é essencialmente a gramática de uma linguagem de programação convencional, mas as dicas gráficas ajudam a tornar evidente como combinar os elementos básicos como variáveis, condições ou controladores de fluxo. Esta linguagem de programação baseia-se na metáfora do bloco de LEGO, onde se encaixam blocos compativeis para criar comportamentos. O Scratch está então muito próximo do código-fonte para uma linguagem imperativa, mas usa blocos visuais o que ajuda na descoberta da sintaxe e evita erros ao tornar óbvio como construir uma instrução e como os elementos se relacionam uns com os outros. Pode ser utilizado em modo offline ou online, descarregando a plataforma para o próprio computador apenas dá para usar em modo offline, enquanto que se for usada a partir do website do Scratch é em modo online, o que permite a partilha de projetos para os restantes utilizadores desta linguagem de programação visual.

3.1.1.1 Interface

A interface do utilizador do Scratch esforça-se para facilitar a navegação. É usada uma *single-window*, com design de vários painéis para garantir que as componentes-chave são sempre visíveis. O Scratch evita janelas flutuantes que podem ficar enterradas. Os objetos animados do Scratch são designados como *Sprites*, sendo estes uma imagem bidimensional que é integrada numa cena maior e que tem associados a si os seus sons, invocados a partir de blocos do Scratch, trajes, que representam a sua aparência no palco, podendo ser alterados, e *Scripts*, blocos de intruções que representam os comportamentos dos objetos. Na figura seguinte é possível observar os vários painéis do Scratch.

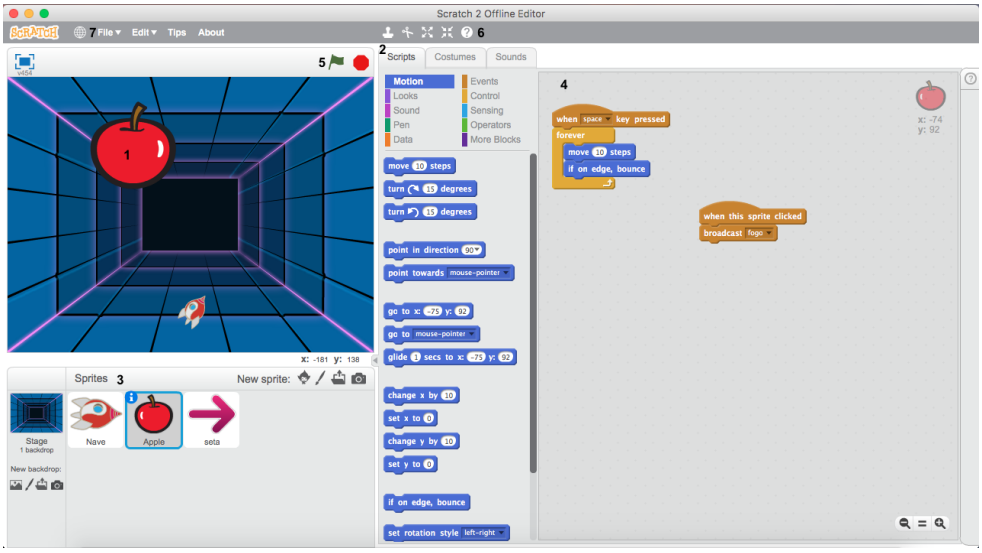


Figura 3.1: Interface do Scratch

1. **Palco ou *Stage*** - Onde toda a acção das aplicações decorre. Onde os *Sprites* se movem e interagem. O cenário de cada palco pode ser alterado para outra imagem.

2. **Separadores de *Scripts, Costumes e Sounds*** - No separador de *Scripts* encontram-se várias categorias de blocos que o utilizador pode usar para programar os seus objetos.

   - ***Motion blocks*** - são os blocos que controlam o movimento e rotação de um *Sprite*.
   - ***Looks blocks*** - blocos que controlam a aparência de um *Sprite*, permitindo mudar os seus trajes.
   - ***Sound blocks*** - blocos que controlam as funções de som do Scratch.
   - ***Pen blocks*** - controlam o desenho no palco através de uma "caneta"e suas características.
   - ***Data blocks*** - dentro desta categoria existem duas subcategorias, os ***Variables blocks***, que são os blocos que armazenam valores e strings em variáveis, e os ***List blocks***, que tratam da gestão das listas.
   - ***Event blocks*** - são os blocos que controlam eventos e desencadeiam os *Scripts*.
   - ***Control blocks*** - controlam os *Scripts*, fornecendo estruturas de controlo como ciclos, condições e iterações.
   - ***Sensing blocks*** - são blocos de detecção, por exemplo, reportam se o *Sprite* tocou numa determinada cor, ou se o mouse esta a tocar no *Sprite*, etc.
   - ***Operators blocks*** - executam funções matemáticas e manipulação de strings.

   Clicando no separador de *Costumes* é possível aceder à área de trajes de um *Sprite*, onde é possível ver os trajes associados ao *Sprite* atualmente selecionado, fazer as alterações que o utilizador pretende ao traje do *Sprite*, bem como criar novos trajes para o *Sprite* selecionado. Selecionando o separador de *Sounds* é possível aceder à área de sons de um determinado *Sprite* ou do *Stage*. Aqui é possível visualizar os sons associados ao *Sprite* ou ao *Stage*, bem como criar novos sons, podendo o utilizador gravar um novo som ou importar um novo som do disco.

3. **Lista de *Sprites*** - Mostra todas as miniaturas de *Sprites* associadas a um projeto. Clicando na devida miniatura de *Sprite* é possível editar os seus *Scripts*, sons e trajes associados. Ao selecionar um *Sprite*, a área de *Scripts* correspondente ao mesmo abre, podendo agora o utilizador visualizar todos os agrupamentos de blocos que esse Sprite ja contém.

4. **Área de *Scripts*** - nesta área são criados os *Scripts* associados ao *Stage* ou a um determinado *Sprite*. Estes *Scripts* são criados arrastando blocos da paleta de blocos e encaixando com outros blocos, criando assim instruções que determinam o comportamento dos *Sprites* e do *Stage*.

5. **Bandeira verde e sinal de STOP** - clicando na bandeira verde o programa é iniciado, ou seja, todos os *Scripts* iniciados pelo bloco "When green flag clicked"começam a sua execução. O botão sinal de STOP faz com que todos os *Scripts* de todos os objetos que estão em execução parem.

6. **Barra de ferramentas** - com o primeiro ícon é possível duplicar *Sprites*, cenários, sons, trajes, blocos e *Scripts*. O ícone seguinte permite apagar todas as coisas mencionadas anteriormente. Os dois ícones seguintes permitem alterar o tamanho dos *Sprites* em palco, um aumenta o seu tamanho e o outro diminui.

3.1.1.2 Objetos

Como foi referido anteriormente, os objetos no Scratch são designados por *Sprites*, que podem ter as suas próprias variáveis e comportamentos, ou seja, *Scripts*. Não existem classes nem herança, logo este modelo de programação é baseado em objetos mas não orientado a objetos. Ao contrário do que acontece nas linguagens orientadas a objetos, como cada *Sprite* tem os seus próprios *Scripts*, o utilizador não tem de procurar comportamentos que foram herdados de outras classes para perceber o comportamento de um determinado *Sprite*, basta apenas olhar para a área de *Scripts* desse *Sprite* para visualizar todos os comportamentos possíveis do mesmo, pois estão todos definidos lá.

3.1.1.3 Variáveis

No Scratch uma variável é basicamente uma caixa com números nela. É um valor variável gravado na memória do Scratch. Esses números podem ser aumentados e diminuídos e feitos para controlar várias partes de um projeto. Contrariamente às variáveis algébricas, que são geralmente desconhecidas, as variáveis no Scratch são simplesmente valores conhecidos. As variáveis podem ser strings ou números. Ao clicar numa variável isolada na área de *Scripts* é exibida uma pequena bolha informando o utilizador do valor dessa mesma variável. As variáveis em Scratch podem ser locais ou globais. Por padrão, quando uma variável é criada é uma variável global. Estas variáveis podem ser lidas e alteradas por qualquer *Sprite* ou *Stage*. As variáveis locais apenas podem ser alteradas pelo seu *Sprite*, contudo podem também ser lidas por outros *Sprites* e pelo *Stage*. Para criar uma variável, na categoria *data blocks*, seleciona-se a opção *make a variable*. Aparecerá uma caixa onde o utilizador terá de escrever o nome que deseja atribuir à variável e por fim seleciona a opção global ou privada, dependendo se quer uma variável global ou local. É possível editar as variáveis através de dois blocos, sendo que um deles é o bloco de atribuição de um valor numérico ou string e o outro apenas incrementa o valor da variável. O primeiro é o bloco "set [variavel] to []"e o segundo bloco é o "change [variavel] by ()". É ainda possível mostrar ou esconder a variável, sendo os blocos respectivamente "show variable [variable]"e "hidden variable [variable]".

3.1.1.4 Listas

Uma lista é uma ferramenta que pode ser usada para armazenar várias informações ao mesmo tempo. Também pode ser definida como uma variável que contém várias outras variáveis. Uma lista consiste em números alinhados com itens. Cada item pode ser recuperado pelo seu número emparelhado. Tal como as variáveis, as listas são criadas e nomeados pelo utilizador, e podem ser também do tipo global ou local. No entanto, uma lista local apenas pode ser editada e lida pelo

Sprite em que foi criada, o que não acontece com as variáveis. Não há limite para o comprimento de um item da lista, nem para a quantidade de itens que uma lista pode ter. A criação das mesmas é feita exactamente da mesma forma que a criação de variáveis. Existem vários blocos para a edição de listas, estes permitem adicionar valores numa determinada posição da lista, ler uma determinada posição, procurar um valor numa lista, mostrar lista, esconder lista, substituir um determinado item por outro à escolha do utilizador, determinar o tamanho da lista, etc.

3.1.1.5 Condições

O Scratch apresenta dois blocos para representar condições. O mais simples, o bloco "if [] then", enquadra comandos a executar se a condição for verdadeira e que não serão executados se a condição for falsa. O outro bloco "if [] then [] else []"pode enquadrar um conjunto de comandos para o caso da condição ser verdadeira e outro conjunto de comandos caso seja falsa. Na figura seguinte são mostrados 2 exemplos da utilização destes blocos.

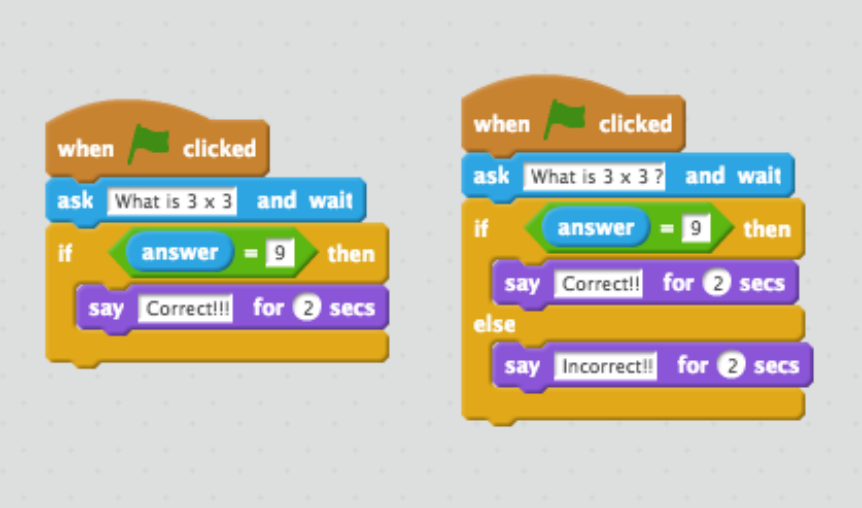


Figura 3.2: Exemplo de condições no Scratch

3.1.1.6 Ciclos

O Scratch apresenta três blocos para executar ciclos. O bloco "*forever*", que é infinito, logo não proporciona uma condição de paragem do ciclo. Outro bloco é o "*repete N times*" que só avança para o bloco seguinte após ter executado as instruções do seu interior o número de vezes que o utilizador definiu e o último é o "*repeat until*" que permite repetir o conjunto de blocos inseridos na sua cavidade até que a sua condição específica seja verdadeira. Na seguinte figura é possível observar exemplos da aplicação destes mesmos blocos.

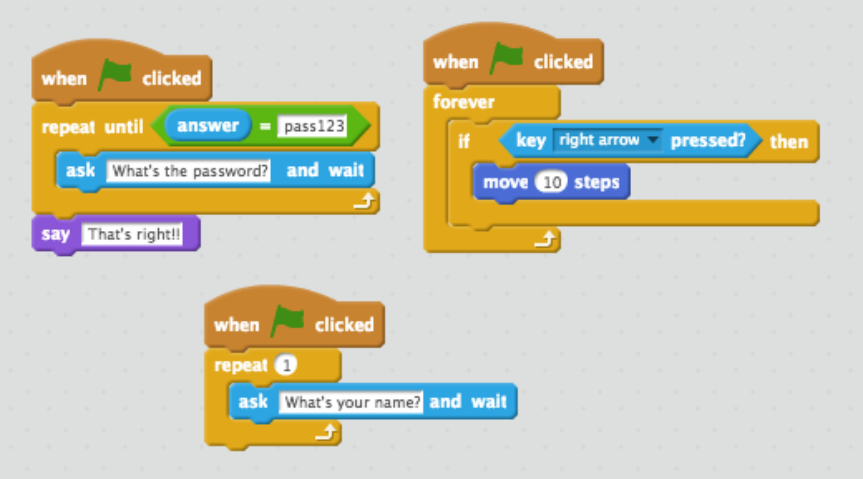


Figura 3.3: Exemplo de ciclos no Scratch

3.1.1.7 Compilação e Execução

Não há nenhuma etápa de compilação, ou seja, não existe distinção entre o modo de edição e o de execução. Os utilizadores podem clicar num comando ou programa a qualquer momento para ver o que ele faz. Podem alterar parâmetros ou adicionar blocos num *Script* enquanto ele está a ser executado. Um bloco pode ser testado clicando nele, mesmo na palete de blocos. Os blocos de função mostram o seu valor de retorno num "balão". Para ajudar os utilizadores a explorar mais facilmente o que os blocos fazem, cada bloco vem com parâmetros padrão que dão uma demonstração do que esse bloco faz. Para além disto é possível usar a ajuda do Scratch para cada comando. Quando a solução dá problemas, um script longo pode ser quebrado em blocos mais pequenos e cada peça testada independentemente. O Scratch fornece feedback visual para mostrar a execução do *Script*. Quando um *Script* está a ser executado, ele é sublinhado por uma borda branca. Com isto os utilizadores conseguem ter feedback sobre quando é que os *Scripts* são executados e quanto tempo demoram a ser executados. Se houver um erro nalgum *Script* o limite do mesmo fica vermelho e o bloco que causou o erro é realçado a vermelho.

3.1.1.8 Debug

O Scratch não apresenta nenhuma ferramente de *debug*, contudo, tanto o facto de os blocos que causam erros ficarem a vermelho, como o modo de *Single-Stepping*, que permite executar os *Scripts* mais rápido ou mais lento e executa cada bloco individualmente, ficando os blocos intermitentes quando estão a ser executados, são funcionalidades que ajudam no controlo dos programas.

3.1.2 Flowgorithm

O Flowgorithm [Coo] é uma ferramenta gratuita de criação gráfica que permite aos utilizadores escrever e executar programas usando flowcharts (tipo de diagrama que representa um algoritmo, fluxo de trabalho ou processo, mostrando as etapas como caixas de vários tipos e a sua ordem, conectando-os com setas). A abordagem é projetada para enfatizar o algoritmo em vez da sintaxe de uma linguagem de programação específica, isto porque programar com esta linguagem passa apenas por arrastar e largar *shapes*, sendo que a sintaxe é reduzida ao mínimo. Por sua vez, o flowchart final pode ser convertido nas seguintes linguagens de programação: C#, C++, Delphi/Pascal, Java, JavaScript, Lua, Perl, Python, QBasic, Ruby, Swift 2, Visual Basic .NET, and Visual Basic for Applications

3.1.2.1 Interface

A interface do utilizador do Flowgorithm apresenta duas janelas, a janela de edição, onde o utilizador elabora os seus programas e a janela de consola, onde são mostrados os resultados dos programas. Na figura seguinte é possível ver a interface desta ferramenta.

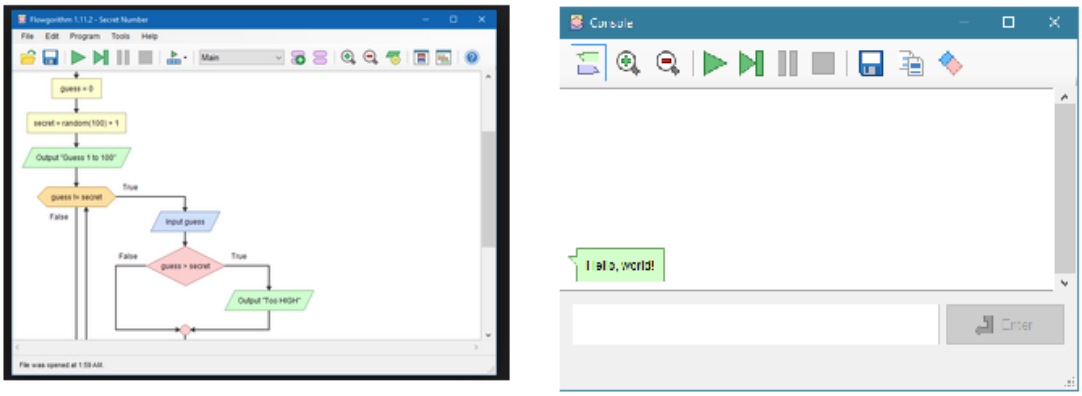


Figura 3.4: Janela de edição do Flowgorithm

Analisando primeiramente a janela de edição, é possível identificar imediatamente abaixo da barra de menu uma barra de ferramentas que contém os comandos usados com mais frequência para gestão de ficheiros e para a execução das operações.

CLicando na primeira opção desta barra, o utilizador pode importar um fluxograma criado e guardado anteriormente. Com o botão seguinte, poderá guardar o seu fluxograma que está a desenvolver. Ao clicar no botão seguinte, o botão de play, o fluxograma será executado. No seguinte botão, o utilizador poderá executar o programa passo por passo. Também é possível, através do botão de pausa, interromper por instantes a execução. Clicando no botão de stop, botão seguinte da barra de ferramentas, a execução pára e termina. Através do próximo item, é possível alterar a execução do fluxograma. A dropdown seguinte serve para o utilizador selecionar funções e o seguinte botão para adicionar. É possível ampliar ou diminuir a vista.

Clicando com o botão direito do rato na janela de edição, todas as *shapes* serão mostradas ao utilizador.

3.1.2.2 Formas gráficas

Quando se inicia um flowchart são mostrados dois rectângulos arredondados chamados de terminais, “main” e “end”, que representam respectivamento o ínicio e o fim de cada programa. Tudo no flowchart é representado por *shapes*, estas estarão sempre entre o “main” e o “end”. Para adicionar uma shape é necessária passar o rato sobre a linha que liga cada *shape* e clicar. Nos flowcharts cada acção que o computador pode tomar é representada por *shapes* diferentes. Na seguinte figura estão representadas as diferentes *shapes* do Flowgorithm.

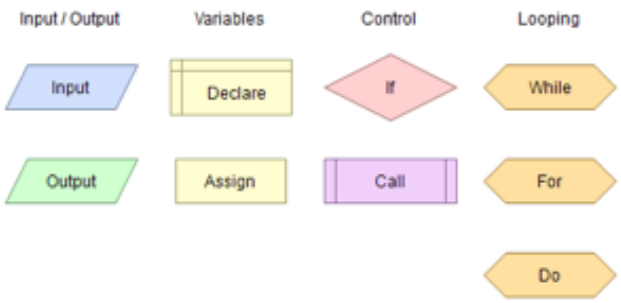


Figura 3.5: Shapes do Flowgorithm

- ***Input* e *Output*** - contem duas shapes, uma para o *input*, utilizado para receber uma variável do teclado e a de **output**, utilizado para apresentar uma mensagem no ecrã; contem duas shapes, uma para o *input*, utilizado para receber uma variável do teclado e a de **output**, utilizado para apresentar uma mensagem no ecrã, ou seja, para exibir uma sequência de caractéres ou para concatenar ou juntar uma string com uma expressão. O Flowgorithm utiliza o simbolo «<e COMERCIAL MUDAR ISTO MAS NAO SEI COMO POR NO LATEX»»> para concatenar;

- ***Variables*** - contem duas *shapes*, uma para declarar variáveis (*Declare*) e outra para executar *assign* de uma expressão a uma variável já declarada (*Assign*);

- ***Control*** - contem duas *shapes* responsáveis pelo controlo do programa, uma para criar uma condição ”if”, (*If*) e outra para chamar uma função criada pelo utilizador, (*Call*);

- ***Looping*** - contem as *shapes* representativas das instruções utilizadas em ciclos, uma para criar um ciclo ”for” (*For*), outra para criar um ciclo ”while”, (*While*) e ainda outra para criar um ciclo ”do”, (*Do*).

3.1.2.3 Variáveis

[Kou16] O Flowgorithm apresenta apenas um tipo de varável, as variáveis locais. Usa então uma declaração estática onde o tipo de dados da variável é determinado no momento da compilação. Cada variável tem de ser declarada antes de ser usada numa expressão através da *shape Declare*. Os tipos de dados que o Flowgorithm define são os seguintes: Inteiro, Real, String e Booleano.

3.1.2.4 Arrays

[Kou16] [UKR] Um array representa uma estrutura básica de dados. Pode ser usado para armazenar um grupo de valores do mesmo tipo (integer, real, string ou boolean), ou seja um array é um grupo de variáveis. No Flowgorithm o índice do array começa com um 0 e deve ser iniciaizado antes de o utilizador tentar aceder ao seu conteúdo. Para aceder aos valores armazenados no array o Flowgorithm utiliza os parênteses retos. Nesta linguagem os arrays são passados por referência por padrão e portanto, a função que recebe o array como seu parâmetro pode alterá-lo.

3.1.2.5 Condições

Na figura seguinte é possível observar o exemplo da utilização "If"no Flowgorithm. Esta instrução verifica uma expressão booleana e em seguida executa o ramo *true* ou *false* com base no resultado. O exemplo seguinte declara um inteiro chamado “idade” através da *shape declare* e de seguida lê a idade do teclado. Finalmente a instrução "If"verifica se a idade é maior ou igual a 18 e com base nisso, se for *false* ele executa esse ramo e exibe “Sorry, not yet” ou executa o ramo *true* e exibe “go vote!”

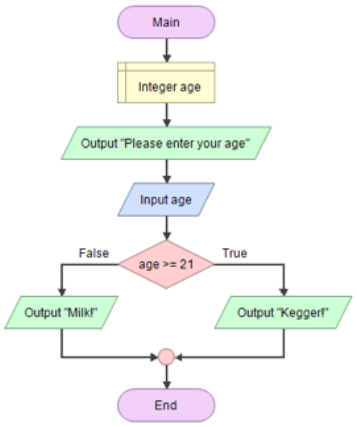


Figura 3.6: Exemplo da condição if no Flowgorithm

3.1.2.6 Ciclos

Como foi dito anteriormente, o Flowgorithm apresenta três *shapes* para executar ciclos. Nas figuras seguintes serão explicadas estes três tipos diferentes de *shapes* recorrendo a exemplos.

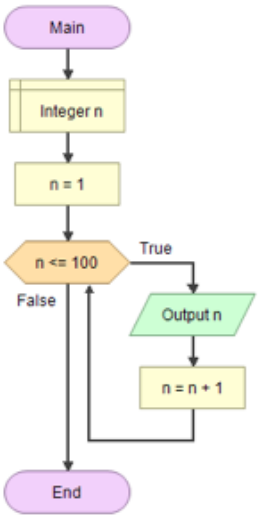


Figura 3.7: Exemplo do ciclo While no Flowgorithm

O cilclo while avalia uma expressão boleana e se for *true* executa as restantes instruções. Ele verifica a expressão váriaa vezes e executa o ciclo até a expressão ser falsa. O exemplo mostra a impressão dos números de 1 a 100. A instrução de atribuição "n=n+1" incrementa a variável "n" em 1 para cada iteração do ciclo.

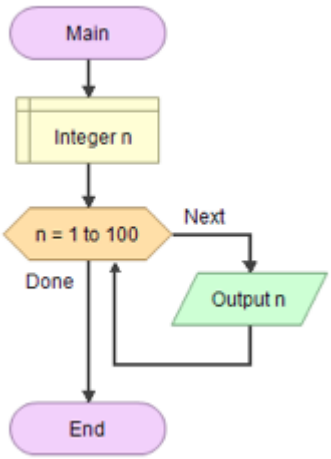


Figura 3.8: Exemplo do ciclo For no Flowgorithm

O Loop For incrementa uma variável através de um intercvalo de valores, sendo uma substituição comum e útil para uma instriçao while. O exemplo faz print de números de 1 até 100. O loop é executado 100 vezes. O valor "n" inicia a 1 e é incrementado por 1 cada vez que o loop é executado. O ciclo termina quando n = 100.

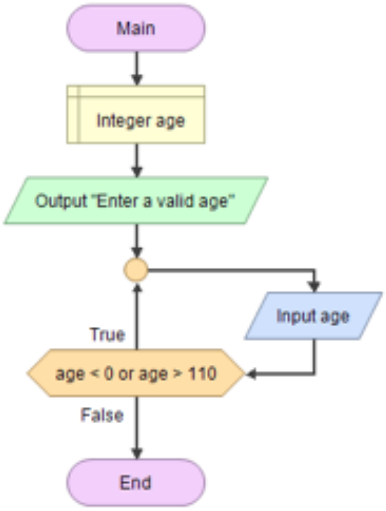


Figura 3.9: Exemplo do ciclo Do no Flowgorithm

O Loop Do é semalhante ao while excepto que o bloco de instruções é executado pelo menos uma vez amntes da expressão ser verificada. O exemplo mostra uma instrução Do que aceita somente a entrada válida. Este loop será executado enquamto que a variável "idade " for menor que 1 ou maior que 100.

3.1.2.7 Funções/Procedimentos

[Kou16] As funções e procedimentos são usados para quebrar uma lógica de programa em componentes gerenciáveis independentes ou interdependentes. Enquanto que uma função tem de retornar um valor usando uma declaração de retorno, um procedimento não retorna valor nenhum. O Flowgorithm permite que o utilizador crie as suas próprias funções, mas não permite que crie um procedimento. No entanto, uma função vazia simula um procedimento. O Flowgorithm não estabelece um limite da quantidade de parâmetros. O parâmetro de suporte no Flowgorithm é compatível com *pass-by-value* para variáveis escalares e *pass-by-reference* para arrays. [oESSP16] Para criar uma função no Flowgorithm é necessário no menu clicar no botão "*Add a Function*"representado na figura a baixo.

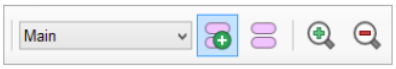


Figura 3.10: Botão *"Add a Function"*

Uma nova janela será aberta onde o utilizador terá de preencher o campo do nome da função, os parâmetros e o valor de retorno da função.

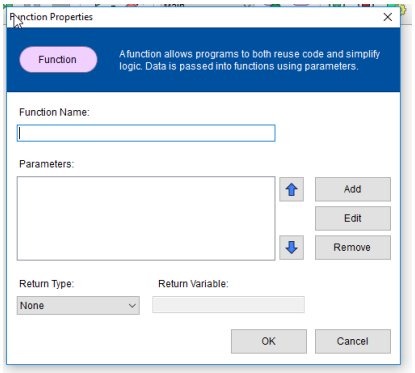


Figura 3.11: Janela de Edição da Função

Depois disto o utilizador terá de acrescentar as *shapes* necessárias à função para a mesma produzir o que ele quer e na função main terá de, com a *shape Call* chamar a respectiva função.

3.1.2.8 Recursividade

O Flowgorithm fornece funções usadas para implementar a recursividade, técnica que consiste em uma função chamar-se a si mesma como subrotina, o que permite que a função seja repetida inúmeras vezes uma vez que é chamada durante a sua execução. [Chr06] Na figura seguinte é mostrado um exemplo de aplicação da recursividade no Flowgorithm, onde é feita a soma dos "N"primeiros números inteiros positivos.

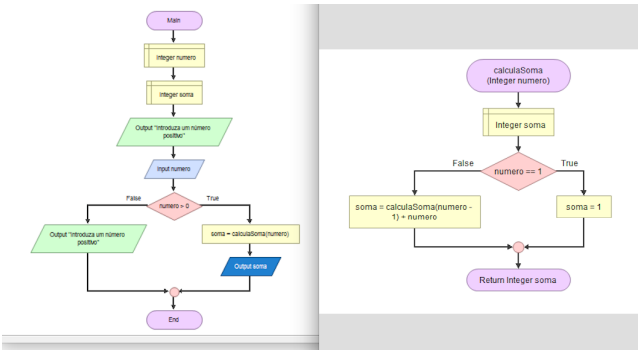


Figura 3.12: Exemplo de Recursividade no Flowgorithm

3.1.2.9 Compilação e Execução

No Flowgorithm não existe qualquer separação entre a compilação e a execução de um programa. Apenas quando é pressionado o botão de "*Run*"uma nova janela é aberta, ja referida anteriormente como sendo a janela de consola , onde o utilizador pode visualizar os resultados

dos seus programas, pausar e parar o programa bem como introduzir os inputs necessários para a execução do mesmo. Tal como no Scratch, no Flowgorithm quando uma *shape* está a ser executada a mesma é realçada a verde, fornecendo assim feedback visual aos utilizadores sobre qual a *shape* em execução e, em caso de erro, qual a *shape* responsável, uma vez que a mesma fica realçada a vermelho. O Flowgorithm apresenta três estados diferentes de execução, *Fast, Medium* e *Slow*, podendo o utilizador selecionar o pretendido aquando da execução do seu programa. As diferenças entre os 3 estados são apenas na velocidade de execução do seu programa, ou seja, o utilizador pode executar os seus programas em velocidade rápida, média ou lenta, podendo com o último modo visualizar mais facilmente quais as *shapes* em execução.

3.1.2.10 Debug

O facto do Flowgorithm apresentar uma funcionalidade de *step-by-step*, ou seja, permitir que a execução do flowchart avance passo a passo com base no fluxo da execução do flowchart, bem como a funcionalidade dos *breakpoints*, que permite parar a execução do flowchart temporariamente num momento escolhido pelo utilizador, são úteis para o debug do programa uma vez que é possível visualizar o conteúdo das variáveis na janela de exibição de variáveis e as shapes que estão a ser executadas, detetando assim muito mais facilmente os erros do programa.

3.1.3 Raptor

O RAPTOR [WCHM] – Rapid Algorithmic Programming Tool for Ordered Reasoning é um ambiente de desenvolvimento de programação visual baseado em flowcharts. Os programas no Raptor são criados visualmente e executados visualmente traçando a sua execução através dos flowcharts. A sintaxe necessária é reduzida ao mínimo. É uma ferramenta open-source que suporta totalmente programação orientada a objetos incluindo encapsulamento, herança e polimorfismo. O flowchart final pode ser convertido em várias linguagens de programação.

3.1.3.1 Inteface

O ambiente de programação do Raptor é composto por duas janelas, as quais podem ser visualizadas na figura seguinte. A janela da esquerda é a janela de edição, ou seja, a janela onde os flowcharts são criados e a da direita é a MasterConsole, janela onde são visualizados os outputs dos programas criados pelos utilizadores.

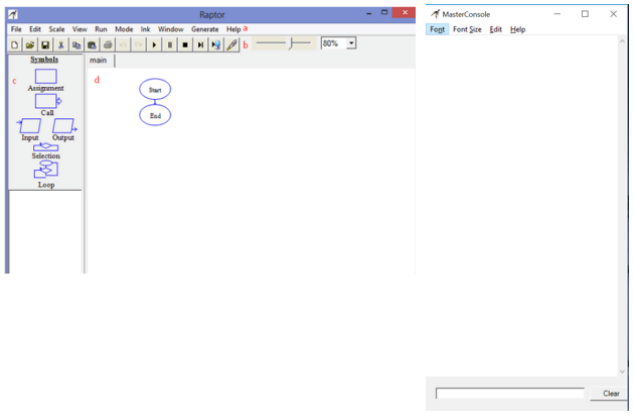


Figura 3.13: Interface do Raptor

Como é possível visualizar na imagem, o painel de edição do Raptor está dividido em 4 secções principais.

(a) Menu:

- **File:** Aqui o utilizador pode executar as seguintes funções, criar um novo flowchart (New), abrir um flowchart previamente guardado (Open), guardar o flowchart actual (Save) e guardá-lo com um nome diferente (Save As), compilar o flowchart (Compile – um flowchart compilado não pode ser visualizado ou editado, apenas é executado, sendo a versão compilada do mesmo muito mais rápida do que a versão gráfica, não podendo no entanto ser executada símbolo por simbolo, ser parada e exibir as variáveis na janela de exibição), personalizar a exibição da página para impressão (Page setup), exibir uma visualização de como o flowchart aparecerá quando impresso (Print Preview), imprimir o flowchart actual (Print), guardar uma imagem bitmap do flowchart actual na área de transferência (Print to Clipboard), sair do programa (Exit) e ainda visualizar uma lista do histórico dos últimos flowcharts acedidos.

- **Edit:** As funções possíveis de realizar são as seguintes: fazer undo, redo, copy, cut, paste, delete, comment – permite ao utlizador adicionar comentários aos símbolos, e ainda select all que permite ao utilizador selecionar o programa todo, ou seja, todo o flowchart para realizar depois as outras ações.

- **Scale:** Aqui o utilizador poderá reajustar a escala da área de trabalho.

- **View:** Aqui o utilizador poderá fazer com que todo o texto de cada símbolo seja exibido completamente (All Text), exibir apenas a quantidade de texto que cabe dentro de cada símbolo (Truncated), esconder o texto de todos os símbolos (No text), alternar a exibição dos comentários no workspace (Comments), alternar a exibição das variáveis na janela de exibição (Variables), expandir todas as estruturas de loops e selections que tinham sido colapsadas (Expand All), colapsar todas as estruturas de loops e selections para minimizar o tamanho do flowchart (Callapse).

3.1.3.2 Variáveis

No raptor, as variáveis são criadas automaticamente assim que são usadas pela primeira vez num símbolo. Com o símbolo Assignent podemos definir uma variável e atribuir lhe um determindo valor. Uma variável pode então ter o seu valor definido (ou alterado) de 3 maneiras, pelo valor inserido a partir de um simbolo de Input, pelo valor calculado a partir de uma equação num símbolo de Assignment, ou por um valor de retorno de uma Procedure Call. Os identificadores do raptor não são sensíveis a maiúsculas e minúsculas. Se dentro de um símbolo está uma seta a apontar para a esquerda então à variável nomeada no lado esquerdo será atribuído o valor do lado direito. Todas as variáveis são do tipo Number ou String , sendo que um número pode ser do tipo inteiro ou um numero decimal que o raptor não faz distinção entre os tipos. Uma string é um texto value entre “”, podem ser frases caracteres palavras ou ate mesmo números. Uma variável deve então receber um nome, um tipo e muitas vezes um valor inicial.

Declaração de variáveis: Quando um programa inicia a sua execução não existem variáveis. A primeira vez que o raptor encontra um novo nome de variável ele cria automaticamente uma nova localização de memoria e associa esse nome de variável à nova memoria. Portanto no raptor não é necessário declarar variáveis explicitamente como se faz noutro tipo de linguagens com o Declare. A variável existirá então a partir do momento em que é criada na execução do programa ate ao seu fim. **Inicialização de variáveis:** Quando uma nova variável é criada no raptor, deve ter um valor inicial, o que pode não acontecer noutras linguagens, onde as variáveis podem ser declaradas sem valor inicial. O valor inicial aqui determina o tipo de variável, se o valor for um numero, a variável é do tipo number, se for um texto, é do tipo string. O tipo de dados de uma variável não pode mudar durante a execução do programa. Enquanto que uma variável no raptor pode ser criada apenas dando lhe um nome e um determinado valor, uma variável não pode contudo ser utilizada sem ter sido criada.

3.1.3.3 Formas Gráficas

Como disse anteriormente um programa em Raptor é um conjunto de símbolos conectados que representam acções a serem executadas. As setas que conectam os símbolos determinam a ordem em que as acções serão então executadas. Ao executar um programa no Raptor começa-se no símbolo **START** e seguem-se as setas para executar. Um programa pára de executar quando o símbolo **END** é atingido. O Raptor apresenta 6 símbolos, onde cada símbolo representa um tipo de instrução. Na figura seguinte é possível visualizá-los.

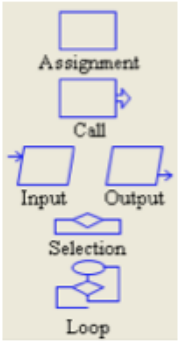


Figura 3.14: Símbolos do Raptor

Input: as linguagens de programação possuem instruções que permitem ao programa obter informações do utilizador através do teclado ou rato e exibir essas informações no ecrã do computador. No raptor quando um simbolo de Input é executado uma janela nova é exibida solicitando ao utilizador que insira um valor. O utilizador insere o valor que é armazenado numa determinada variável. Quando o utilizador insere um simbolo de input no flowchart é necessário passar lhe alguma informação, ao clicar duas vezes no simbolo uma nova janela ira aparecer onde o utilizador terá de preencher dois campos. Na parte "*Enter Input*" da janela escreve uma sequência de texto que descreve a entrada que será necessária, devendo este texto ser o mais explícito possível, na parte "*Enter variable here*" deverá inserir a variável onde o utilizador quer que as informações fornecidas pelo utilizador em tempo de execução fiquem armazenadas, pode-se inserir o nome de uma variável já declarada ou inserir outro pois a variável é automaticamente criada. Para a parte do *"Enter input"* para além de texto, a expressão também pode ter variáveis, tendo que o texto estar entre "" e as variáveis nao, e sendo que a ligação entre ambos tem de ter um +. Quando o programa está a correr e encontra o símbolo de input uma janela aparecerá para o utilizador preencher com o determinado valor.

Assignment: é usado para definir o valor de uma variável ou para executar uma expressão matemática, sendo depois os resultados armazenados numa variável. Estes valores armazenados podem ser recuperados e usados em *statements* mais tarde. O Raptor primeiro processa a expressão do lado direito da seta do assignment, depois esse valor é colocado no local de memória associado à variável do lado esquerdo substituindo qualquer valor que tenha sido armazenado anteriormente. A expressão de um símbolo *Assignment* pode ser uma simples ou complexa equação resultando no fim num valor único. Se a variável do lado esquerdo da seta não existir, a mesma é criada automaticamente. Nenhuma variável do lado direito da seta, ou seja, da expressão, é alterada pela símbolo *assignment*.

Procedure Call: um procedimento é um conjunto de instruções de programação que realizam uma determinada tarefa. A chamada de um procedimento suspende a execução do programa, executa as instruções no procedimento chamado e, em seguida, retoma a execução do programa. Para a utilização de um procedimento é necessário saber-se o nome do procedimento e os valores que o procedimento necessita, que são chamados de argumentos. Para minimizar o número de

nomes de procedimentos que o utilizador tem de memorizar, o raptor na janela "Enter Call" exibe todos os nomes dos procedimentos, esta janela também ajuda dizendo quais os argumentos que esse procedimento necessita.

Output: exibe um valor para a *MasterConsole* quando executado. Tal como no Input, aqui também uma janela nova aparecerá, onde é pedido ao utilizador que especifique o texto ou expressão que queira como output. É possível por texto, variáveis ou uma combinação de texto e variáveis.

Estruturas de Controlo: As estruturas de controlo permitem que o utilizador determine a ordem em que as instruções do programa são executadas, permitem que algumas instruções não sejam executadas ao executar outras e permitem que algumas instruções sejam repetidas enquanto uma determinada instrução for verdadeira. O raptor tem dois tipos de símbolos considerados estruturas de controlo, as ***Selections*** e os ***Loops***. Nos símbolos vistos anteriormente o controlo do programa era sequencial, ou seja era sempre executado um símbolo após o outro, ou seja, o utilizador coloca cada simbolo na ordem que deseja e o programa corre da instrução Start para a instrução End.

Controlo de seleção: uma instrução de controlo de seleção permite que o utilizador faça decisões no seu código sobre o estado actual dos dados do seu programa e depois selecione um dos dois caminhos alternativos para a próxima instrução/simbolo. Na figura a baixo pode-se visualizar um exemplo de um programa com a intruçao ***Selection***. Todas as decisões são declaradas como "sim/não", quando um programa é executado se a a condição da *selection* for verdadeira, o ramo da esquerda será executado, se for falsa será o ramo da direita executado. É possível criar expressões para a instrução *selection* com os seguintes operadores: =; !=; /=; <; <=; >; >=; and; or; xor; not. Os primeiros operadores devem sempre comparar dois valores do mesmo tipo de dados. Este tipo de símbolo/instrução é semelhante a um If numa linguagem de programação clássica. Quando é necessário fazer uma escolha entre mais do que duas opções são necessárias mais instruções de seleção, quando isto acontece é referido que é um controle de seleção em cascata.

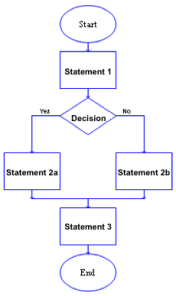


Figura 3.15: Controlo de Seleção

Loops: uma instrução de loop, em várias linguagens de programação, permite que o utilizador repita uma ou mais instruções até que alguma condição se torne verdadeira. No raptor o loop é representado por uma elipse e por um losango, como é possível ver na imagem a baixo. O número de vezes que o loop é executado é controlado pela expressão de decisão, que é inserida no losango.

Durante a execução, quando o simbolo do losango é executado, se a expressão de decisão, ou seja, a condição do utilizador, for avaliada como "não/falsa", então o ramo não é executado, o que leva de volta à instrução de loop e repetição. As instruções que o utilizador quer que sejam repetidas podem ser colocadas acima ou abaixo do losango de decisão.

Uma utilização da instrução ***loop*** que se pode comparar com a instrução ***For*** numa outra linguagem de programação é o ***Counter-controlled loop***. Este *loop* executa um pedaço de código um número especifico de vezes , requer uma variável que é incrementada em cada execução do loop, o contador, que é inicializada antes do loop, incrementada dentro do loop e é usada na expressão de decisão para parar o loop. Exemplo:

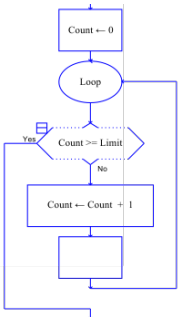


Figura 3.16: Loop

3.1.3.4 Modos de Utilização

O Raptor apresenta três modos de utilização, o modo ***Novice***, que é o modo default do Raptor e o mais simples também, onde todas as variáveis são variáveis globais independentemente do flowchart em que forem declaradas, o modo ***Intermediate***, que permite ao utilizador criar procedimentos que tenham o seu próprio domínio, introduzindo assim os conceitos de passagem de parâmetros e suporte à recursividade e a possibilidade de declarar variáveis locais, e o modo ***Object-Oriented***, que, como o próprio nome indica, é apropriado para a criação de programas orientados a objetos, onde o utilizador pode criar classes com métodos e atributos e instanciar objetos. Neste último modo de utilização é adicionado um novo símbolo de intrução, o *Return*.

Mode Object-Oriented: Com este último modo o utilizador terá duas tabs de edição, a tab de edição main (janela principal, janela esta utilizada nos outros dois modos) e a tab UML, que é usada para criar a estrutura de um programa orientado a objectos. As classes são então criadas na tab UML. A imagem seguinte mostra a tab UML.

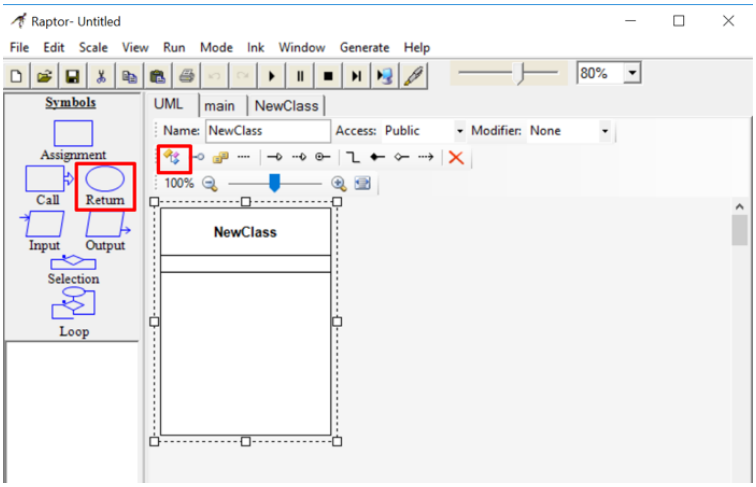


Figura 3.17: Tab UML

Para adicionar uma classe é necessário clicar no botao destacado a vermelho. É de notar também que um novo símbolo é adicionado à palete de símbolos nesta tab, o símbolo Return. Clicando então no botão Add New Class destacado na imagem, uma classe nova é adicionada à janela como é possível observar. Clicando duas vezes com o rato sobre a classe é possível adicionar novos membros, funções e atributos. No Raptor os atributos são chamados de Fields (campo). Na figura a baixo é possível observar uma nova janela que abre para permitir ao utilizador adicionar os membros, onde é possível optar por adicionar uma função ou um field. Um field deve receber um data type, enquanto que numa função, se receber um valor passado pelo main então os utilizadores terão de incluir esse parâmetro. Por exemplo: se estivermos a calcular o volume de um cubo a função SetSide() está a passar o valor do comprimento de um lado e por isso a sintax seria a seguinte: public void SetSide(int NewSide); outro exemplo, a função ComputeVolume(), que servirá para calcular o volume do cubo, usa o valor do lado do cubo para os seus cálculos e por isso necessita de um parâmetro, a variável inteira Side, public void ComputeVolume(int Side); a função GetSide() como não necessita de nenhum parâmetro terá a seguinte sintax public void GetSide().

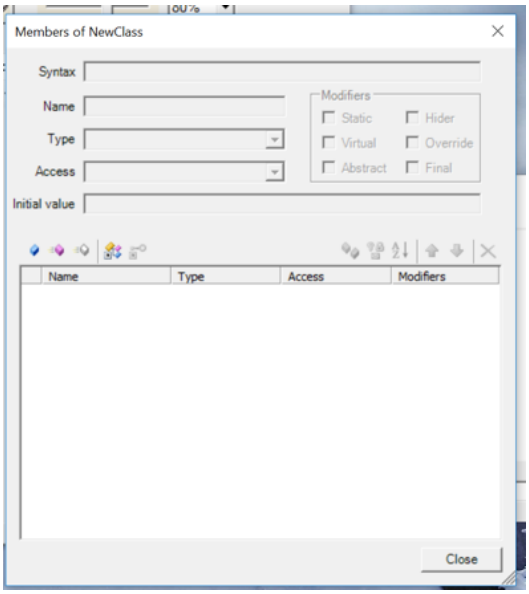


Figura 3.18: Janela da Nova Class

Assim que a classe é criada uma nova tab será automaticamente adicionada com o nome da classe. Agora é necessário criar o código de cada função de cada classe. Clicando na tab da classe vemos as novas tabs, uma para cada método. Na tab de cada função e com o auxilio dos símbolos será feita a programação das mesmas, recorrendo sempre a flowcharts. Após isto já é possível fazer a programação do main, que irá como é de esperar chamar as funções criadas previamente. Para criar uma classe filho é necessário carregar no botão Inherits from , depois clicar na classe filho e só depois na classe main.

3.2 Domínio de Jogos

3.2.1 Kodu

O Kodu [Cor] é uma linguagem de programação visual desenvolvida especificamente para a criação de jogos. Esta linguagem foi projetada para ser acessível para crianças e divertida para qualquer utilizador. O ambiente de programação é executado na Xbox e também em Microsoft Windows. Ao contrário de outras linguagens de programação, o Kodu é totalmente orientado a eventos, pelo que a programação envolve a colocação de *tiles* numa sequência para formar uma condição e ação em cada regra. Além da gestão dos objectos (personagens do Kodu), é também possível criar um mundo em 3 dimensões com vários elementos. Os objectos são programados individualmente decidindo-se quando é que uma acção vai ser iniciada e qual será essa ação. A programação em Kodu baseia-se portanto em condições e ações. Cada linha de programação está sob a forma de uma condição e uma acção, referida como uma regra. Cada palavra na regra é representada como uma peça e portanto é um membro do alfabeto desta linguagem de programação.

3.2.1.1 Interface

A interface gráfica do utilizador facilita o desenvolvimento do jogo através do uso de uma coleção de ferramentas no Menu Ferramenta.

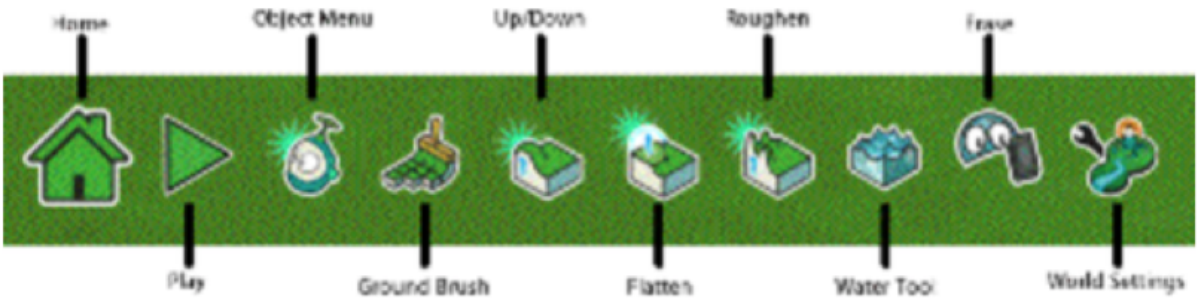


Figura 3.19: Menu Ferramenta

Como é possível observar na imagem, este menu inclui: opções para adicionar, editar e programar objetos dentro do programa (Object Tool); Três ferramentas de edição de terreno; Uma ferramenta de edição de água; Uma ferramenta de delete; E uma ferramenta geral de configurações mundiais. Há também uma opção de jogo e um ícone de casa, que permite o acesso ao menu inicial. Muitas opções apresentadas ao programador, incluindo o processo de seleção de um objeto para adicionar ao mundo, são feitas através do fornecimento de um círculo de seleção representado na figura a baixo.

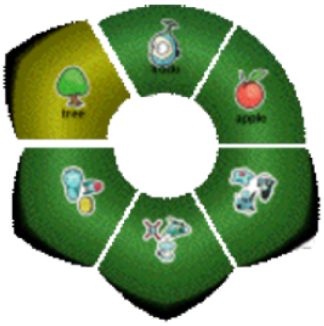


Figura 3.20: Círculo de Seleção

Editores de Terreno: Para construir o mundo do jogo, o kodu inclui três ferramentas de edição de terreno e uma ferramenta de água, com a qual o utilizador pode criar o seu próprio mundo e ambiente. O programador pode selecionar de uma escala extensiva de texturas do terreno (22 escolhas). As ferramentas de edição do terreno incluem uma escova de terra para criar ou editar terra, uma ferramenta de terra achatada para tornar o solo liso ou nivelado, e uma ferramenta de

terreno áspero para a criação de solo cravado ou montanhoso. A ferramenta de água permite que o programador inclua água no mundo do jogo, definindo o nível de água e selecionando a cor (10 opções).

Tiles: Os utilizadores programam o seu mundo selecionando opções da roda de seletor e a tile relevante é então inserida na linha de código. Dependendo da declaração que está a ser feita, certas tiles podem ser incluídas na instrução when ou do. Alguns blocos podem ser incluídos em qualquer uma das declarações when ou do.

3.2.1.2 Ambiente de Programação

Os passos genéricos para fazer um programa em kodu são os seguintes:

- criar um mundo

- adicionar personagens ao mundo

- programar as personagens que se adicionam, ou seja, programar o seu comportamento perante determinados acontecimentos

- correr o programa.

O mundo do kodu pode incluir objetos, variáveis, sons, caminhos (paths) e um ambiente.

3.2.1.3 Objetos

Como referi anteriormente, cada personagem no Kodu é um objeto, e cada uma possui variáveis locais. Um utilizador pode criar novos objectos , excluir objectos e clonar objectos, podendo estes ser criados e removidos durante o tempo de execução. Na maior parte dos casos, um clone executa uma cópia profunda incluindo o código e as propriedades do objecto inicial. No entanto, se o objecto for declarado como criável, então um clone será um novo objeto que faz referência ao original e imita o seu comportamento. Quando um objecto com clones é modificado as alterações à sua programação afetam todos os seus clones. Os objetos disponíveis no KGL incluem as personagens e os elementos do ambiente (árvores, maçãs, rochas, moedas e nuvens) que são programáveis. A extensão da programabilidade de cada objeto depende do seu tipo e dos seus atributos inerentes. A principal diferença entre os objectos e os elementos do ambiente é que um objecto pode ser programada para se mover, enquanto que a maioria dos elementos de ambiente não se podem mover. Dentro do mundo do jogo cada objeto pode ver e ouvir todos os outros objetos (a menos que definido como invisível), e pode detetar quando outro objeto colidiu com ele.

3.2.1.4 Variáveis

Simplificando, uma variável é um símbolo que contém um valor. No Kodu como noutras linguagens, as variáveis podem ser locais ou globais.

Globais: As variáveis globais no Kodu podem ser lidas ou escritas por qualquer personagem. No Kodu, os scores, que possuem valores inteiros, atuam como variáveis globais. Existem 37 pontuações globais que podem ser mantidas por qualquer programa, uma para cada letra do alfabeto e cor suportada pelo Kodu.

Locais: As personagens têm quatro propriedades locais que são semelhantes a variáveis locais. Estas propriedades são a cor, brilho, expressão e vida. A cor, o brilho e a expressão guardam um valor (por exemplo, laranja, feliz) e a vida contém um valor inteiro. Uma personagem pode ler e escrever todas as propriedades para si e algumas das propriedades de outras personagens. Por exemplo, kodu pode ser programado para alterar a propriedade de vida de um ciclo, mas não pode verificar a o valor da vida do ciclo.

Random: Uma maneira comum de introduzir o não determinismo na programação é através de variáveis aleatórias, e o Kodu adotou essa prática. Existem três blocos de variáveis aleatórias que podem ser usados, um para o tempo, outro para a pontuação e um para a cor. Estes podem ser usados como filtros ou modificadores para desencadear o comportamento ou definir uma propriedade ou valor aleatoriamente.

3.2.1.5 Condições

Como foi referido anteriormente, a programação no Kodu baseia-se em condições e respectivas ações. Cada linha de programação está então sob a forma de uma condição e uma acção, referindo assim uma regra. Um exemplo de uma regra poderia ser: "quando vê uma maçã vermelha, avançar rapidamente", aqui, quando vê a maçã vermelha é a condição e o avançar rapidamente é a acção. Cada palavra na regra é representada como uma peça e portanto é um membro do alfabeto. Esta descrição de linguagem é representada por uma série de regras de produção, onde o lado esquerdo (LHS left hand side) mostra uma variável, também conhecida como não-terminal, e o lado direito (RHS right hand side) contém variáveis e terminais. Cada terminal é um elemento no alfabeto da linguagem Kodu e todos começam com uma letra minúscula. No caso de Kodu, o alfabeto é composto de todo o conjunto de tiles disponíveis durante a programação. Os programas são organizados como páginas de regras e todas as regras de uma página aplicam-se simultaneamente. Assim, a maneira mais geral para uma personagem mudar o seu comportamento é mudar para outra página.

3.2.1.6 Compilação e Execução

Para executar um programa apenas é necessário o utilizador sair do menu de programação, clicando na tecla "Esc"e clicar novamente na mesma tecla para correr o programa.

3.2.2 Cryengine Flowgraph

[CRY] Flow Graph é um sistema de script visual incorporado no CryENGINE Sandbox Editor. A principal vantagem do Flow Graph Editor é que os utilizadores não precisam ter nenhum script ou conhecimento de programação. A sua lógica simples e também complexa pode ser construída

apenas com cliques e sem exigir qualquer script ou codigo. Para além de ser a principal ferramenta usada para criar missões de lógica ao nível de single players, também pode ser usado para protótipos de jogabilidade, efeitos e design de som. Os níveis podem ter múltiplos gráficos executando diferentes tarefas ao mesmo tempo. Este sistema da aos designers uma interface intuitiva para criar e controlar eventos, triggers, lógica do jogo, efeitos e desenho de som.

No Flow Graph cada nó é um elemento lógico (flow component), onde cada componente possui um nº de entradas e saídas, todas as entradas (excepto Events e any) contêm valores de porta padrão que podem ser modificados com o editor, como tal as portas de entrada também servem de propriedades das componentes. Assim que uma componente recebe um valor ou um evento numa determinada entrada a mesma é activada. Este método verifica as portas que são realmente ativadas e recupera o valor da porta de entrada e executa a ação específica do nó. Uma componente também pode ativar arbitrariamente qualquer uma das suas saídas, se esta saída estiver conectada à porta de entrada de outra componente, evento de output ou valor, irá imediatamente propagar o link e ativará a porta de entrada da próxima componente. A saída de qualquer componente de fluxo pode ser conectada à entrada de outro componente de fluxo com um link direcionado. As saídas e entradas são classificadas com base no tipo de dados, que podem ser: boolean value, floats, integer, string, vector; cada uma representa uma cor, azul, branco, vermelho, turquesa e amarelo respectivamente. Quando o output de um determinado tipo está ligado a um input de outro tipo há uma conversão de tipo no valor de output.

O Flow Graph usa nós para representar entidades ou comportamentos que podem ser controlados vinculando-os a outros nós. Os nós podem ser interligados por links:

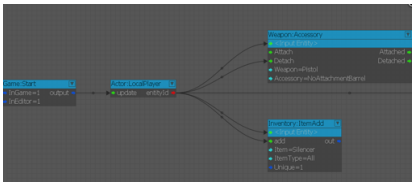


Figura 3.21: Links

A lógica do Flow Graph é armazenada no formato XML e pode ser facilmente exportada para ser usada nos outros níveis. Como um gráfico é sempre criado e armazenado numa entidade específica, o gráfico é sempre exportado juntamente com o objeto. As camadas são totalmente suportadas no sistema Flow Graph.

3.2.2.1 Terminologia

Grafico: Um único flow graph é referido como um gráfico **Nós**: Os nós são a representação de entidades (nó entidade) ou componentes (nó componente) que executam certas operações. Todos os nós têm inputs (info que entra) e outputs (info que sai). **Nó Componente**: é um nó que não representa uma entidade real do nível, mas executa uma ação específica. **Nó Entidade:** representam as entidades do nível. As portas de entrada e saída dependem das portas definidas na

entidade. **Links**: conectam os nos. São visualizados como linhas desenhadas entre as portas dos nos conectados **Portas**: Os nós têm portas de entrada e saída, essas portas são usadas como uma conexão para links de outros nos. **Entidade gráfica:** entidade que contem um gráfico, é o objecto que seleccionamos no cryengine sandbox. É necessário existir uma entidade no cryengine para se criar um flowgraph e assim programar esta entidade.

3.2.2.2 Interface

O Flow Graph é uma ferramenta integrada no Cryengine que é usada para controlar os eventos e a lógica de jogo dentro dos níveis. A janela desta ferramenta está dividida em 9 partes diferentes, como é possível ver na imagem.

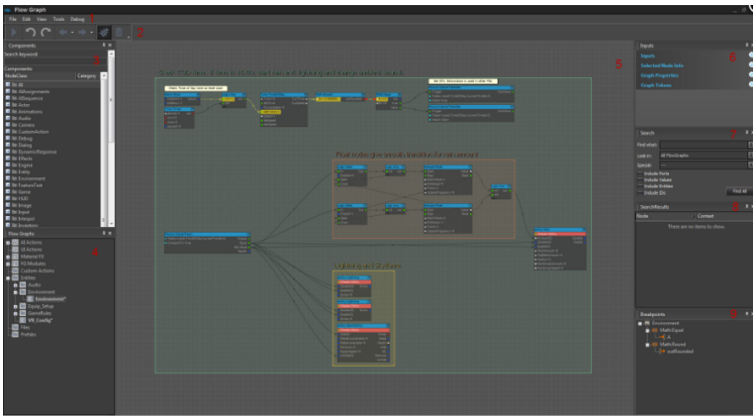


Figura 3.22: Janela de Edição

Barra de Menu: 1. File – aqui é possível fazer as seguintes acçoes: abrir um novo gráfico de fluxo vazio; criar açoes AI, UI e personalizadas; criar um novo módulo de FG, Global pode ser compartilhado entre níveis, Level, só pode ser visto e usado num determinado nível; Salvar, Importar e Exportar (*.xml) 2. Edit – é possível realizar ações como, Undo, Redo, Copy, Paste, Cut, Paste with links (enquanto que o paste apenas copia os nós, esta ação copia também os links associados aos nós copiados), Delete e Find (vai para a janela de search). 3. View - Esta janela permite ocultar / mostrar todas as janelas diferentes desta ferramenta. 4. Tools – é possível editar os Tokens e os Módulos do Flow Graph (mais à frente tenho a explicação do que representam os Tokens e os Módulos) 5. Debug – Ativar Debug (destaca o fluxo lógico dos gráficos, o destaque é representado pelos destaques amarelos nos links), Apagar informação de Debug (limpa os destaques amarelos nos links) e Ignorar o tipo de Flow Graph.

Barra de Ferramentas: barra de ferramentos de acesso rápido. Tem as açoes de Undo e Redo, mais as opções de debug vistas anteriormente, a opção de previous e next que avança para o flow graph seguinte ou para o anterior respectivamente e a opção de Start Flow graph update, que faz com que o jogo retome após um breakpoint ter sido atingido no modo debug.

Janela de Componentes: esta janela é dividida em duas secções: a. Campo de pesquisa - aqui é possível pesquisar os nós a partir de qualquer parte do seu nome, filtrando na lista de nós todos

os que tenham esse texto no nome b. Lista de nós – nesta lista todos os nós do flow graph podem ser encontrados, como disse anteriormente, os nós de componente são nós que não representam uma entidade do nível, mas possuem uma funcionalidade abstracta que pode usar uma ou mais entidades Adicionar nós ao gráfico: Os nós são classificados por categorias e podem ser adicionados ao gráfico de fluxo atual de 3 maneiras: Arrastando um nó da lista de componentes e soltando-o no painel de edição; clicando no botão direito do mouse no painel de edição e selecionando um nó no ponto do menu AddNode no menu de contexto; ou na janela principal do Flow graph, pressionar a tecla Q abrindo assim uma lista rápida popup, depois digitamos o nome do nó que procuramos e a lista de nós é reduzida o máximo que puder. As categorias a que os nós podem pertencer são as seguintes: • Approved: apresentam funcionalidades simples e são portasnto mais acessíveis de usar • Advanced: apresentam funcionalidades mais complexas e por isso sao aconselhados a serem usados por utilizadores mais experientes • Debug: nós utilizados para fins de debbug • Legacy: • WorkInProgress: • No Category:

Lista de Flow Graphs: esta janela fornece uma visão geral dos diferentes gráficos e entidades. As entidades só podem ter 1 gráfico anexado a elas, não múltiplo.

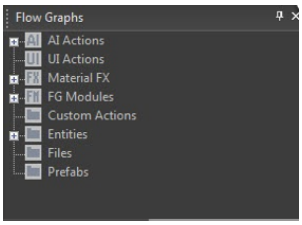


Figura 3.23: Lista de Flow Graphs

Janela Principal: onde toda a edição ocorre. Os nós podem ser adicionados, excluídos, movidos e vinculados aqui. Esta janela precisa de muito espaço de tela para funcionar efetivamente. É possível navegar e estender o painel de edição em qualquer direcção. É possível selecionar e desmarcar os nós aqui presentes. Clicando duas vezes num nó será selecionada a entidade associada ao nó no nível. Clicando com o botao direito do rato um menu de contexto será exibido, onde serão possíveis fazer as seguintes funções: adicionar nó, adicionar uma entidade selecionada na janela de exibição 3D, adicionar comentários (simples – texto escrito directamente no flow graph, caixa de comentários – agrupar elementos chave no flow graph ou black box – para esconder nós da vista principal do flow graph) , adicionar track event node, adicionar o start node, copy, cut, paste, paste com links, delete, selection (abre um sub menu onde se podem selecionar as seguintes acçoes: exportar nós selecionados - exibe os nós selecionados para uma folha * .xml a ser importada para outro gráfico de fluxo; selecionar entidade associada – seleciona e destaca a entidade associada na janela principal do viewport; seleciona e vai para a entidade associada), importar – importa um flow graph externo a partir de um ficheiro *.xml, show spline arrows, ajustar o gráfico para visualizar – automaticamente reduz o flow graph para que seja possível visualiza-lo.

Janela de Inputs: A janela Inputs é específica para o nó atualmente selecionado. Todos

os parâmetros do nó podem ser editados aqui. Além dos controles de entrada, há uma guia de informações que fornece uma descrição do nó atualmente selecionado.

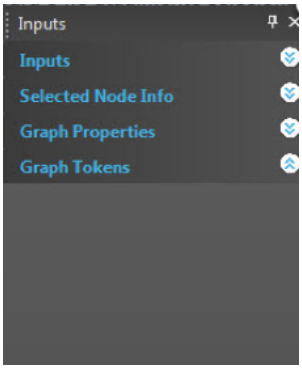


Figura 3.24: Janela de Inputs

Janela de Search: Os flow graphs e gráficos de acção podem ser pesquisados por nós e / ou valores específicos. As opções de pesquisa podem ser configuradas para incluir ou excluir diferentes partes dos nós. A pesquisa também pode ser limitada a gráficos de ação ou entidades. A opção Look no drop-down define quais os gráficos que serão pesquisados: gráfico atual; AIActions; Entities; All Flowgraphs A opção Special no drop-down oferce mais opções para excluir certas categorias de nós

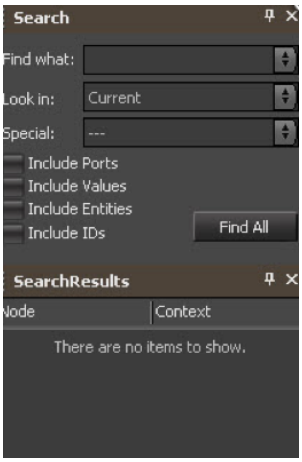


Figura 3.25: Janela de Search

Janela de SearchResults: aqui serão exibidos todos os resultados da pesquisa e clicando duas vezes irá para o gráfico correspondente.

Janela de BreakPoints: esta janela mostra os breakpoints que definimos para fazer o debug de um determinado flow graph. Os breakpoints podem ser adicionados às portas de entrada e saída dos nós. Estes serão representados no gráfico como uma bola vermelha ao lado da porta em que foi adicionado. Uma vez que o breakpoint foi desencadeado pelo jogo, o jogo irá parar até

ser aprovado pressionando F5 , ou clicando na seta para continuar. Para adicionar breakpoints é necessário carregar na tecla Ctrl + LMB, sobre a porta de entrada/saída. Para desativar o breakpoint é exactamente da mesma forma. É possível remover tambem através da janela de breakpoints selecionando o que queremos eliminar ou selecionando o pai principal r assim remover todos os breakpoints. Assim que o breakpoint é activado, o jogo pára ate haver algum input do utilizador.

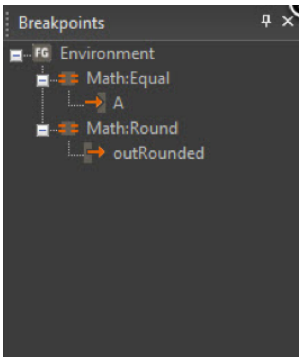


Figura 3.26: Janela de break points

3.2.2.3 Adicionar e Editar Nos:

Adicionar e editar nós: Os nós entidade actuam sempre numa instância específica de uma entidade no nível. Os nós componentes são independentes de entidades e usam as entidades como um alvo no qual executam determinadas ações. O alvo de um nó pode ser reatribuído e alterado sempre que necessário.

Para adicionar um nó referente a uma entidade, basta selecionar a entidade pretendida e depois, já na janela do flow graph com o botão direito do rato selecionar o seguinte, como é possível ver na imagem, adicionando assim um nó ao objecto que foi selecionado previamente e programando assim o objeto/entidade selecionada. Para adicionar um nó componente é como adicionar um nó entidade, mas sem entidade selecionada, ou seja é aberta a janela que vemos na imagem e depois clicar em add node.

Os nós podem ser editados de duas maneiras: através da janela nó, no lado direito do editor do flow graph usando a tab de Inputs, ou editando os parâmetros directamente nos nós.

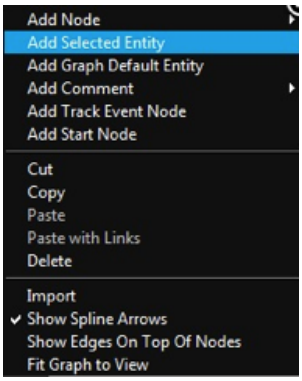


Figura 3.27: Adicionar Nós

3.2.2.4 Gerir e Editar Links:

Criar e editar Links: Para criar um novo link entre dois nós, clica se em qualquer porta de saída e arrasta-se para a porta de entrada desejada. Para excluir um link, clica-se com o botão direito do rato no link e selecionamos a opção Remover.

Cada link possui um meno de contexto que tem 4 açoes possíveis: Remover, Desactivar, Ativar e Time (O nó de atraso fornece um atraso entre os nós conectados pelo link com um tempo de atraso padrão de 1 segundo). Cada porta de entrada de um nó pode apenas ter um link conectado a ela, enquanto que as portas de saída dos nós podem ter vários links.

Os links de entrada e saída são destacados para facilitar o debug de gráficos mais complexos. É possível visualizar o tipo de links conectados ao nó com base na cor, os links de entrada são destacados a vermelho e os links de saída a azul.

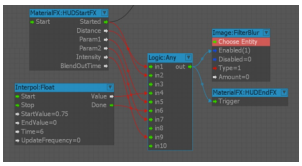


Figura 3.28: Edição de links

Como disse anteriormente, cada porta de entrada de um nó apenas tem um link conectado a ela, mas é possível através do uso do nó any, um nó auxiliar, actuar como uma ponte para conectar mais do que um link a uma porta de entrada, o que é possível visualizar na figura a cima.

3.2.2.5 Gerir Flowgraphs:

Como disse anteriormente, estes gráficos pertencem a uma entidade específica sendo então armazenados como propriedade da entidade. Quando a entidade é salva ou exportada o flowgraph correspondente é automaticamente guardado. Para criar um flowgraph é necessário selecionr a entidade na viewport e depois ir para a janela de flowgraphs e na tab de propriedades há 3 opçoes,

abrir, listar ou remover. Se for o 1º flowgraph desse nível será necessário inseri lo num grupo, senão é so selecionar o grupo a que queremos que pertença. Para eliminar um flowgraph, como o mesmo está associado a uma entidade, se a entidade for eliminada o flowgraph associado será também eliminado. É possível alterar o nome do gráfico e da entidade, como estão associados, se um deles for alterado o outro é imediatamente alterado também pois o nome do gráfico é o nome da entidade. Para ativar ou desativar os gráficos, ao clicar com o botão direito do rato no gráfico desejado na janela de visão geral do Gráfico de fluxo e selecionar-se a opção Desativar, o mesmo será desactivado, assim, os nós dentro dos gráficos serão ignorados enquanto o jogo estiver a ser executado. É possível depois voltar a ativá-lo selecionando a opção ativar no menu.

3.2.2.6 Module System:

Um Flowgraph Module é uma maneira de encapsular nós como o seu próprio gráfico que assim pode ser chamado a partir de outros gráficos quando estão a ser executados. Para criar um novo modulo é necessário ir a file, e selecionar a opção new module, aqui teremos a opçaõ de criar um modulo global (para todos os níveis) ou level (apenas para um nível específico). Para chamar um módulo de outro gráfico é necessário usar os nós Call Nodes. Cada módulo tem o seu próprio nome de Call Node: Module:Call_<Module Name>. É possível chamar os módulos por call nodes diferentes e a partir de gráficos diferentes. Cada chamada ao módulo vai gerar uma instância separada que corre independentemente da chamada anterior.

3.2.2.7 Debug:

Como referi anteriormente, é possível adicionar breakpoints em qualquer porta de entrada e saída de um determinado nó. Assim qe o jogo é parado devido ao breakpoint, é aberto o respectivo nó do breakpoint. Para ativer o debug é necessário clicar no botao de debug na barra de ferramentas. Quando o breakpoint é adicionado ao nó um ponto vermelho é adicionado à porta do nó onde foi adicionado. É possível ativar e desativar breakpoints específicos. É possível converter todos os breakpoints em tracepoints, ao contrário dos breakpoints estes não pausam o jogo e a info é mostrada directamente na consola.

3.3 Domínio de Multimédia

3.3.1 Quartz Composer

[Qua] O Quartz Composer é uma ferramenta de desenvolvimento para processar e interpretar dados gráficos. O ambiente de programação visual permite desenvolver módulos de processamento gráfico, chamados composições, sem escrever uma única linha de código. O Quartz Composer é também uma framework que permite aceder, gerir e manipular programaticamente composições criadas com a ferramenta de desenvolvimento.

3.3.1.1 Elementos do Quartz Composer

Composições: As composições Quartz são programas de motion graphics criados pela montagem de módulos preexistentes (chamados patches) num workflow para processamento e interpretação de dados. Podem ter parâmetros de entrada e produzir resultados de saída.

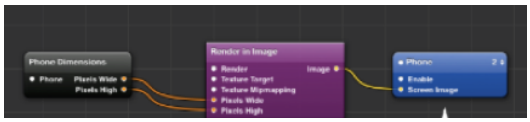


Figura 3.29: Composições

Patches Os elementos básicos do Quartz Composer são patches . São semelhantes a rotinas em linguagens de programação tradicionais, os patches são unidades de processamento base. Eles executam e produzem um resultado. São entidades visuais. Os círculos em um patch representam portas , com portas de entrada no lado esquerdo de um patch e portas de saída no lado direito. As portas passam dados através deles. As conexões entre as portas definem como os dados fluem quando a composição é executada.

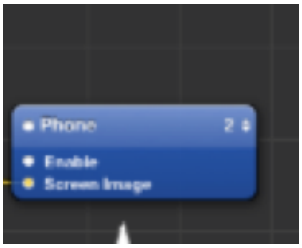


Figura 3.30: Patches

Modos de execução dos patches Há 3 tipos de patches que definem o seu modo de execução, consumidor – passa um resultado para o destino, processador - processa dados em intervalos especificados ou em resposta a alterar os valores de entrada, ou provedor - fornece dados de uma fonte externa para uma composição. As cores dos patches indicam o seu modo de execução. Os processadores são pretos, os fornecedores são roxos e os consumidores são azuis

3.3.1.2 Interface:

Janela de Editor: A parte principal da janela do editor do Quartz Composer é a área de trabalho - uma grade para montar e conectar patches. O botão Patch Creator abre a janela do Patch Creator onde se pode procurar nomes e categorias de patches, ou pode usar o campo de pesquisa para procurar patches

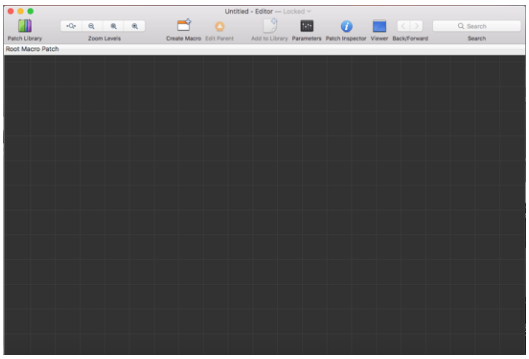


Figura 3.31: Interface Quartz

Patch parameters: O botão Patch Parameters na barra de ferramentas da janela do editor alterna o painel de parâmetros de patch.

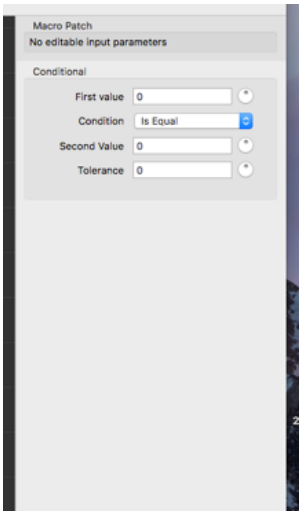


Figura 3.32: Patch Parameters

Patch Inspector: tem 3 paineis possíveis. Na primeira imagem esta o painel dos parâmetros de entrada, onde é possível editar os parâmetros de entrada dos patches, na segunda está o painel de settings e na terceira o painel de as portas de entrada para um patch macro

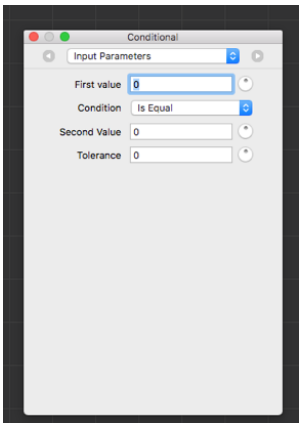


Figura 3.33: Parâmetros de Entrada

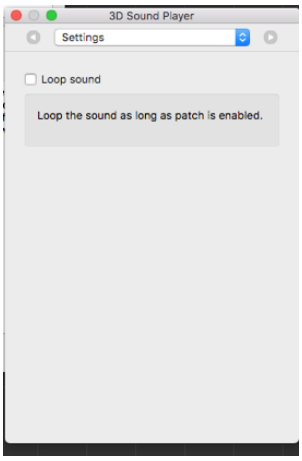


Figura 3.34: Painel de Settings

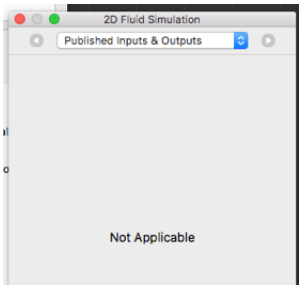


Figura 3.35: Painel de Portas de Entrada

Janela de Visualizador A janela de visualizador tem sua própria barra de ferramentas. Pode-se iniciar e parar a renderig, alternar para rendering em tela cheia, alterar o modo de rendering, visualizar parâmetros de entrada de composição e alternar para a janela do editor.

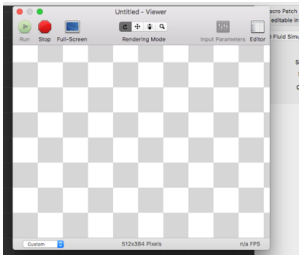


Figura 3.36: Janela de Visualizador

3.3.1.3 Modos de Rendering

Performance: é o modo de rendering normal e fornece um desempenho ideal. Ao ativar o modo de tela cheia, o Quartz Composer usa sempre o modo de rendering de performance.

Perfil: reúne estatísticas exibidas numa janela à direita da janela do visualizador

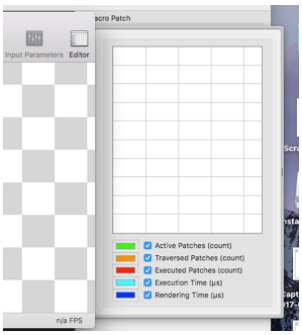


Figura 3.37: Perfil

Debug: exibe informações de debug exibidas nujma janla abaixo da janela do visualizador

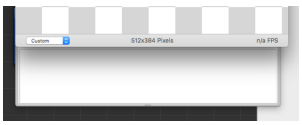


Figura 3.38: Debug

Adicionar Patches Patch Library > arrastar patch ou clicar no patch pretendido Em vez de adicionar um patch Image Importer, pode-se arrastar uma imagem do Finder diretamente para o espaço de trabalho. O Quartz Composer cria automaticamente uma instância do patch Image Importer e usa o nome da imagem como título do patch. Compositor Quartz fornece uma conveniência semelhante para filmes. Em vez de adicionar um patch do Movie Loader, basta arrastar um movimento QuickTime do Finder para o espaço de trabalho.

3.3.2 Blender Nodes

[Ble] O blender é uma ferramenta open-source de criação 3D, pode ser usado para criar visualizações 3D como imagens estáticas, vídeos e jogos interactivos em tempo real. O Blender apresenta diferentes editores, entre os quais o Blender nodes, que permitem a modificação e exibição dos diferentes aspectos dos dados nos projetos.

Falando agora sobre o Blender node Editor, este é usado para trabalhar em fluxos de trabalho com base na passagem das imagens e valores através de nós. Com este editor é possível construir 3 tipos de árvores de nós, dependendo do tipo de nós, existem os nós de material , nós de composição , e nos de textura , sobre os quais irei falar mais à frente. O Blender nodes editor é uma linguagem visual que faz , no caso dos materiais, a ligação entre objectos, materiais, podendo criar um material com varias propriedades, vários shaders e depois atribuir a um objeto, para alem disto também permite ligar scripts que o utilizador cria em python, pois vem com um interpretados de python. Nestes scripts é possível fazer tudo, interagir com objetos, modifica-los, fazer materiais, criar menus e GUI. Não é possível fazer ciclos, condições, etc.. pois o Blender nodes é limitado, dá apenas para criar conjunts de nos já predefinidos e fazer a ligação entre eles. O utilizador pode é usar um script pythin e usar o script node para fazer interações com ele.

3.3.2.1 Interface

Menu: Como é possível visualizar na imagem temos as seguintes opções no menu:

Figura 3.39: Menu Blender

View: Esta tab muda a visualização di editor Select: aqui o utilizador pode selecionar um nó ou conjuntos de nós. Add: A sua função é permitir ao utilizador adicionar nós Node: sevre para fazer varias operaç^{o}oes com os nós selecionados assim como com os vértices. Seguem se os botoes dos nós de Material, Composição e Textura. Como referi anteriormente, no Blender nodes editor os nós estao agrupados Segundo 3 categorias específicas. Use Nodes: Avisa o motor de renderização para utilizar o mapa de nós na computação das cores do material ou na renderização da imagem final, ou não. Caso não, o mapa é ignorado e a renderização básica definida nos painéis de contexto de material ou cena será realizada. Use Pinned: Quando possível, o editor irá reter o material ou textura sendo apresentada, mesmo quando o utilizadoe selecionar um objeto diferente. Uma árvore de nós poderá então ser editada independente da seleção do objeto dentro da Janela de visualização 3D. Go to Parent button: este botao permite ao utilizador ir para o nó pai.

3.3.2.2 Componentes dos Nós:

Todos os nós têm por base uma construção semelhante, seja de que tipo forem.

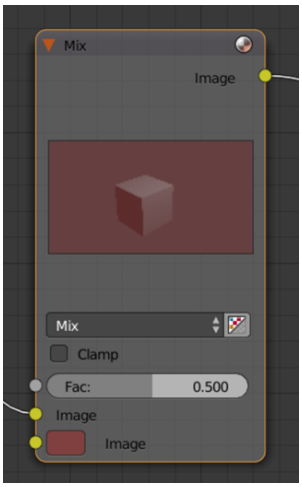


Figura 3.40: Nós

No Blender editor nodes, todos os nós apresentam um título, que, quando o nó está minimizado é das únicas informações que o utilizador consegue visualizar. O titulo representa o nome ou o tipo do nó. Os sockets ou conectores, são as os pontos do lado esquerdo e direito de um nó que fazem a ligação entre os nós, fazendo estes com que os valores entrem e saiam de cada no para nó através de links. Podemos distinguir então dois tipos de conectores, os de input e os de output. Os sockets têm varias cores que representam a sua função, o amarelo serve para a Cor, indica que a informação de cor necessita ser a entrada ou então a saída a partir do nó, o cinza é para entrada ou saída de valores numéricos que tanto pode ser um valor numérico único ou um mapa de valores, o azul representea entrada e saída de Vetores, ou seja, indica informações de vetores, coordenadas e normais, verde está associado ao sombreador que é usado para os sombreadores no Motor de renderização Cycles. Inputs: As Entradas estão localizadas na parte esquerda dos nós, e fornecem os dados que o nó necessita para executar as suas funções. Cada um dos conectores de entrada, exceto a entrada de sombreador na cor verde, quando desconectadas, possuem um valor padrão que pode ser editado através de uma cor, valor numérico ou entrada de vetor por alguma outra interface

Outputs: As Saídas estão localizadas na parte direita dos nós, e podem ser conectadas as entradas dos nós que estão sequenciados, seguindo o fluxo da árvore de nós.

Para alem disto, muitos dos nós possuem propriedades que podem afectar a forma como eles interagem com os outputs e inputs.

É possível selecionar os nós que o utilizadror quer de varias maneiras, dependendo do que o mesmo pretende fazer.

Para adicionar um no o utilizador pode faze-lo de duas maneiras. Usando ferramenta Shelf que possui botões para adicionar nos , estando estes organizados em abas. Ou usando a opção add do menu Shift-A.

Conectar os sockets: com o botao esquerdo do rato clicar num socket e arrastar, saindo assim uma linha do socket que é denominada de link, que tem de ser conectado a outro socket de outro

no. Os sockets podem ter mais do que um link atribuído se forem um socket de output, se forem de input apenas podem ter um único link associado. Os nós que não possuem conexões podem ser inseridos num link, o utilizador apenas tem de mover o no sobre o link e soltá-lo assim que este estiver laranja.

Desconectar sockets: para desconectar sockeyts apenas é necessário arrastar o li nk de um soquete de entrada e deixa lo quando estiver desconectado. Para quebrar um link é necessário clicar em ctrl + botao esquerdo do rato sobre uma área vazia perto do link desejado e arrastar, o utilizador verá um pequeno ícone de corte que aparecerá no ponteiro do rato, depois mve esse ponteiro sobre o link.

Duplicar nos: é possível duplicar nós no Blender editor nodes, para tal , o utilizador terá de clicar com um botao do rato no no desejado e pressionar shoift-D.

Para apagar um no apenas é necessário carregar no botao delete quando o mesmo estiver selecionada.

É possível tabem dar mute a um nó, o que faz é que a contribuição do no para a arvore desse no é removida, ou seja , todos os links passam por esse no e não sofrem qualquer alteração.

3.3.2.3 Grupos de Nós:

Agrupar os nos no Blender nodes pode simplificar a arvore de nos permitindo instancias e esconder partes da arvore, apenas os nos de material e composição é que podem ser agrupados. O agrupamento de nos permite então que o utilizador especifique um conj de nos podendo assim trata-lo como apenas um no. Estes grupos são semelhantes às funções em programação uma vez que o utilizador pode reutiliza-los no grupo ou em outros ficheiros blend chamando se NodeTrees. Por exemplo, se o utilizador criou um material que pretende usar diferentes inputs, como cor, verde e vermelho, o utilizador pode criar diferentes materiais com Make single user para cada cor diferente com uma copia da parte da arvore. Se o utilizador pretendesse editar o material seria necessário refazer a edição em todos os materiais, assim , usando o grupo de nos expondo apenas as entradas variáveis. É possível tameb criar um grupo de nos dentro de outro grupo de nos. Os grupos de nos recursivos são proibidos para todos os sistemas de nos para evitar a recursividade infinita. Um grupo de no nunca se pode conter a si mesmo ou conter outro grupo que o contem. Para criar um grupo de nos o utilizador tem de selecionar os nos que pretende incluir no grupo e em seguida pressionar ctrl-G, group make group. Ao adicionar grupos de nós de um arquivo para outro, o Blender não faz uma distinção entre grupos de nó de material ou grupos de nó de composição.

Nós de composição: este tipo de nos permitem aos utilizadores a montagem e melhoria de uma imagem (ou filme). Utilizando os nós de composição, o utilizador pode colar duas peças de filmagem entre si e colorir toda a sequência num único passo. O utilizador pode melhorar as cores de uma única imagem ou de um filme como um todo de uma maneira estática ou dinâmica que pode ser alterada num determinado período de tempo (conforme o clipe de filme avança). Desta maneira, podem-se usar os nós de composição tanto para montar clipes de filme em conjunto como para melhorá-los. Aqui o termo imagem pode referir-se a uma única imagem, a uma imagem

numa sequencia numerada de imagens ou a um frame de um clip de vídeo. Para uma imagem ser alterada, o utilizador tem então que importar a imagem para o Blender, muda-la, opcionalmente, fundi-la com outras imagens e por ultimo guarda-la. Ex:

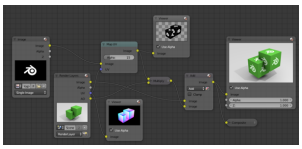


Figura 3.41: Exemplo

Nós de Input:os nos de input produzem informação de alguma fonte, um input poderia ser por exemplo tirado diretamente da camara ativa em alguma cena selecionada. Estes nos geram informação que alimenta outros nos, logo não têm conectores de entrada, apenas de saída **Nós de Output:** são usadas para a saída de resultados **Color Nodes:** ajustam as cores das imagens, por exemplo aumentando o contraste, sobrepondo outras imagens, etc... **Converter Nodes:** convertem as cores ou outras propriedades de diversos dados (por exemplo, transparências) **Filter Nodes:**processam os pixeis das imagens para realçar detalhes adicionais ou perfazer algum tipo de efeito de pós-processamento na imagem. **Vector Nodes:**podem ser usados para manipular diversos tipos de vetores, como as normais das superfícies e vetores de velocidade **No Matte:** Estes nós fornecem a você as ferramentas essenciais para a criação de uma Tela de projeção para imagens que não possuem seu próprio Canal alfa, canal adicional dentro de uma imagem para as transparencias **No Distort:** distorcem as imagens, operando tanto uniformemente na imagem, ou usando uma máscara para variar o efeito sobre a imagem **Nós de Layout:**ajudam a controlar o esquema organizacional e a conectividade dos nós dentro do compositor.

3.3.2.4 Nós de Textura:

este tipo de nos permitem a criação de texturas combinando cores, padroes e outras texturas. Estas texturas podem então ser usadas nos mesmo sítios que as texturas regulares, podem ser colocadas em canais de textura, em nos de materiais, em sistemas de partículas e dentro de outras texturas. Dentro deste conjunto de nos temos os seguintes tipos: Color nodes; Converter Nodes; Distort Nodes; Input Nodes;Output Nodes; Pattern Nodes; Texture Nodes

3.3.2.5 Nós de materiais:

o Blender permite qe o utilizador crie um material ligando materiais básicos através de um conjunto de nós. Cada nó executa alguma operação no material mudando a forma como ele aparecerá quando aplicado à malha e passando para o próximo no. Assim será possível fazer aparências de materiais bastante complexas. Tipos de nos: Nós de cor; Nós de conversão; Nós de input; Nos de Output ; Nós de vetor

Capítulo 4

Inquéritos

Neste capítulo será apresentado o inquérito efectuado bem como os resultados obtidos pelo menos. Na secção 4.1 será descrista a amostra, estrutura do inquérito e escalas usadas. Por fim, na secção 4.2 serão expostos os resultados obtidos e conclusões.

4.1 Descrição da Amostra e Inquérito

Para este inquérito, foi selecionada uma amostra de docentes e alunos que já possuem conhecimento e experiência na área das *Visual Programming Languages*. A amostra consiste em 39 pessoas sendo que 14 são do domínio Multimédia, 14 do domínio de Jogos e 11 do domínio Educacional.

O inquérito consiste em 14 perguntas, sendo o inquérito organizado da seguinte forma: as 2 primeiras perguntas são meramente exploratórias e têm por objetivo conhecer o grau de conhecimento e de uso das linguagens estudadas e outras se for o caso; as 12 perguntas seguintes são de âmbito geral relativo às *Visual Programming Languages*, pelo que os resultados para os vários domínios serão avaliados em conjunto na subsecção 4.2.1; as restantes são específicas a cada domínio e serão avaliadas nas subsecções 4.2.2, 4.2.3 e 4.2.4. As respostas a este inquérito estão estruturadas sob a forma de uma escala de Likert [Ber07]. Assim sendo as respostas a este inquérito são dadas através da seleção de um valor entre 1 e 5, sendo que 1 significa discordo totalmente e 5 concordo totalmente com a afirmação.

4.2 Resultados Obtidos

4.2.1 Perguntas Gerais

Nesta secção são expostos os resultados dos inquéritos relativos às perguntas gerais a todos os domínios. Primeiramente, será apresentada uma tabela síntese onde cada linha da mesma corresponde a cada pergunta do inquérito realizado com as respetivas respostas, média e desvio padrão

das respostas, para que possam ser tiradas as devidas conclusões. Depois da tabela síntese e da respetiva análise aos resultados da mesma, serão expostos os gráficos correspondentes a cada pergunta do inquérito, estando estes em percentagem.

	1	2	3	4	5	Média	Desvio Padrão	Coeficiente de Variação
Considero este tipo de linguagens imprescindível para realizar o meu trabalho na área.	1	8	9	16	5	3,41	1,03	30,23%
Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências bascias de programação para utilizadores iniciantes.	0	3	5	15	16	4,13	0,91	22,06%
Considero que este tipo de linguagens permite aos utilizadores ter uma curva de aprendizagem menor em relação às linguagens de programação convencionais.	0	0	8	18	13	4,13	0,72	17,50%
Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária menor.	4	8	10	8	9	3,26	1,30	39,78%
Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (+18).	4	12	20	3	0	2,56	0,78	30,33%
Considero que este tipo de linguagens pode ser aplicado num ambiente profissional/empresarial.	2	2	4	17	14	4,00	1,06	26,56%
Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico.	0	1	0	18	20	4,46	0,63	14,22%
Considero que este tipo de linguagens permite uma fácil interpretação, de modo visual, do workflow do programa, tanto para o utilizador como para terceiros.	0	1	4	22	12	4,15	0,70	16,84%
Considero que este tipo de linguagens permite detetar facilmente erros/bugs/inconsistências no programa.	0	11	21	6	1	2,92	0,73	24,97%
Considero que com este tipo de linguagens é possível ter um leque de funcionalidades equiparável a uma linguagem de programação convencional.	1	8	9	15	6	3,44	1,06	30,77%
Considero que este tipo de linguagens é bastante intuitivo.	0	1	4	26	8	4,05	0,64	15,76%

Tabela 4.1: Tabela síntese de resultados do inquérito relativo às perguntas gerais a todos os domínios

Após a análise da tabela síntese apresentada em cima e olhando primeiramente para os valores da média de resposta a cada pergunta, é possível perceber que, do ponto de vista dos utilizadores que responderam a este inquérito, e tendo estes já alguma experiência com linguagens de programação visuais, este tipo de linguagens é considerado pelos mesmos de mais fácil aprendizagem do que as linguagens de programação convencionais, uma vez que a média de resposta a esta afirmação foi de 4,13, valor este que corresponde à resposta "Concordo". Também é de salientar que face à afirmação "Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico."a média de resposta foi de 4,46, o que corresponde às respostas "Concordo/Concordo Fortemente". Estando estas duas afirmações relacionadas, uma vez que ambas tratam do tema de aprendizagem de linguagens de programação, uma possível conclusão a retirar desta análise é que, para estes utilizadores, do ponto de vista da aprendizagem de programação, este tipo de linguagens poderá ser uma boa abordagem, pois é também da opinião dos utilizadores inquiridos, que este tipo de linguagens permite uma fácil interpretação, de modo visual, do workflow de um programa. Contudo, face à afirmação "Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (+18)", a média da resposta foi bastante mais negativa, tendo esta um valor de 2,56, o que corresponde à resposta "Discordo/Não concordo nem discordo". Por outro lado, relativamente à afirmação "Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes"a média de resposta já foi de 4,13, correspondendo à resposta "Concordo". Com a análise das respostas a estas duas últimas afirmações é possível reforçar a conclusão inicialmente tirada, para os utilizadores inquiridos, este tipo de linguagens são de mais fácil aprendizagem do que as linguagens de programação convencionais e por isso seriam uma boa aposta para utilizar em jovens sem bases de programação, mas, tendo em consideração as restantes respostas, conclui-se que os inquiridos consideram que apesar de serem linguagens intuitivas, poderão não ser tão poderosas como as linguagens de programação convencionais, pois a maior parte dos utilizadores respondeu que não é fácil detetar erros e bugs ao longo de um programa e que o leque de funcionalidades destas linguagens não é equiparável com as funcionalidades de uma linguagem de programação convencional. Analisando agora os valores obtidos no cálculo do desvio padrão assim como no coeficiente de variação, de uma primeira análise é possível concluir que em bastantes perguntas as opiniões dos utilizadores inquiridos divergiu um pouco. Tendo como valor de referência 20% para o coeficiente de variação em que, quando um coeficiente de variação é superior a 20% trata-se de uma amostra heterogénea e quando o coeficiente de variação é inferior a 20% trata-se de uma amostra homogénea, é possível referir que em 7 das 11 perguntas do inquérito, a dispersão das respostas foi bastante grande. O que pode levar a conclusões positivas ou negativas relativamente à utilização ou não deste tipo de linguagens. Observando primeiramente a afirmação 2, é possível referir que a opinião dos utilizadores divergiu um pouco face ao valor da média. Nesta afirmação, como mencionado anteriormente, a média de resposta foi de 4,13, ou seja, em média os utilizadores consideraram que estas linguagens são uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes, no entanto, o valor do desvio padrão foi de 0,91 e o coeficiente de variação de 22,06% pois apesar da média ter sido 4,13, houve

16 inquiridos que responderam "Concordo Fortemente", que corresponde a um valor de 5 e 8 com um valor inferior a 4, o que fez aumentar o valor do coeficiente de variação. Contudo, mais de metade dos inquiridos concorda com a afirmação. Por outro lado, relativamente à afirmação número 3 o valor do coeficiente de variação foi inferior a 20%, o que torna as conclusões retiradas mais exatas e precisas pois a opinião dos utilizadores nesta afirmação não divergiu muito, sendo então possível afirmar com toda a certeza que as pessoas consideram que a curva de aprendizagem deste tipo de linguagens é menor relativamente às linguagens convencionais. Um valor que se encontra bastante destacado é o valor do coeficiente de variação relativo à pergunta número 4. Face a este valor, 39,78%, as conclusões a retirar não podem ser muito concretas, isto porque as respostas dos utilizadores foram bastante dispersas, tendo também um valor de média de resposta de 3,26, que corresponde a "Não concordo nem discordo". Relativamente à afirmação "Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico", os valores obtidos no desvio padrão e no coeficiente de variação foram de 0,63 e 14,21 respetivamente, o que permite concluir que as respostas dos utilizadores não divergiram muito e por isso concordar totalmente com a afirmação. Por fim analisando a última afirmação, é possível concluir que o valor da média é um valor que se aproxima bastante da opinião de todos os utilizadores inquiridos, pois o coeficiente de variação foi de 15,76.

A seguir são apresentados todos os esquemas relativos a cada pergunta do inquérito analisado. Os resultados dos esquemas estão em percentagem.

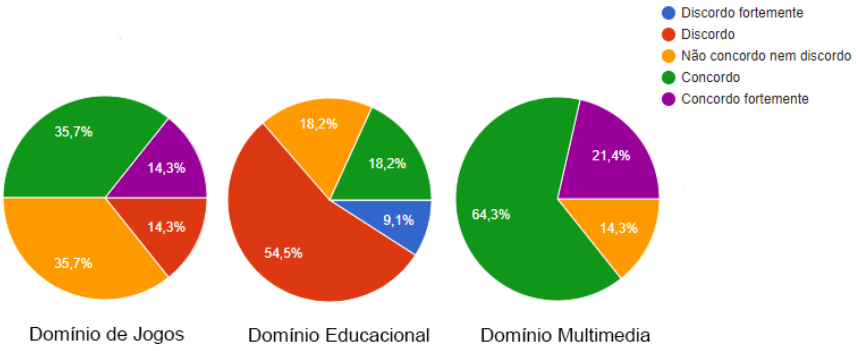


Figura 4.1: Considero este tipo de linguagens imprescindível para realizar o meu trabalho na área.

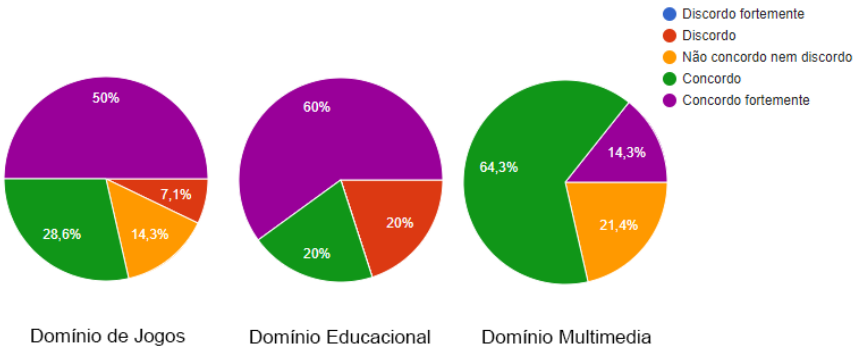


Figura 4.2: Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes.

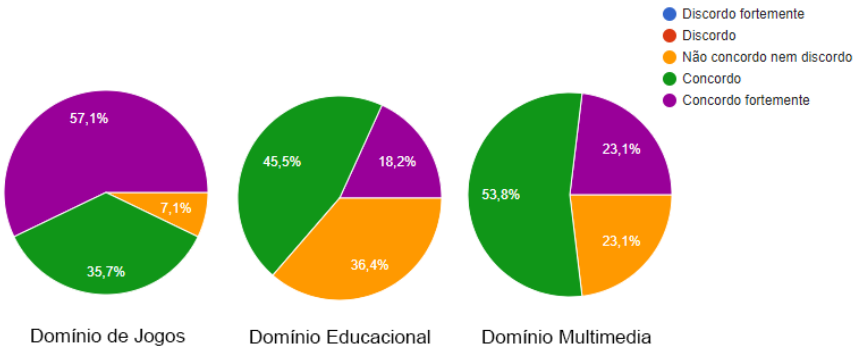


Figura 4.3: Considero que este tipo de linguagens permite aos utilizadores ter uma curva de aprendizagem menor em relação às linguagens de programação convencionais.

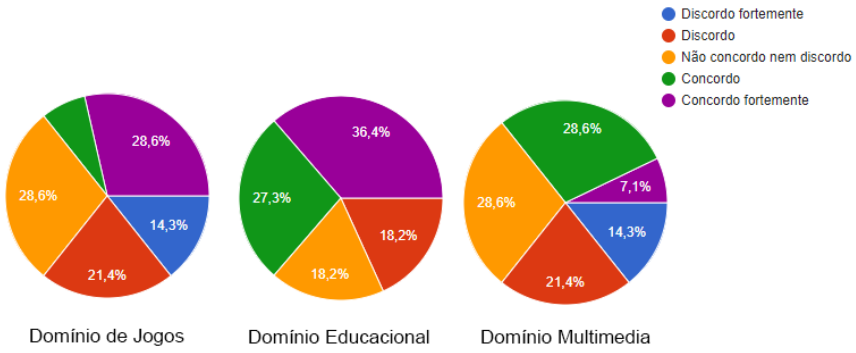


Figura 4.4: Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária menor.

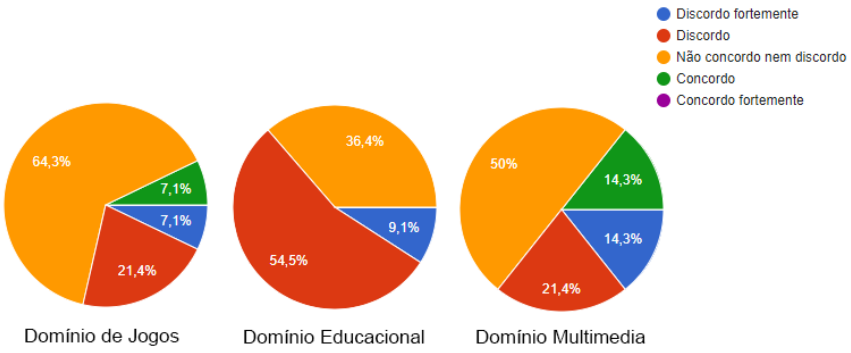


Figura 4.5: Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (18+).

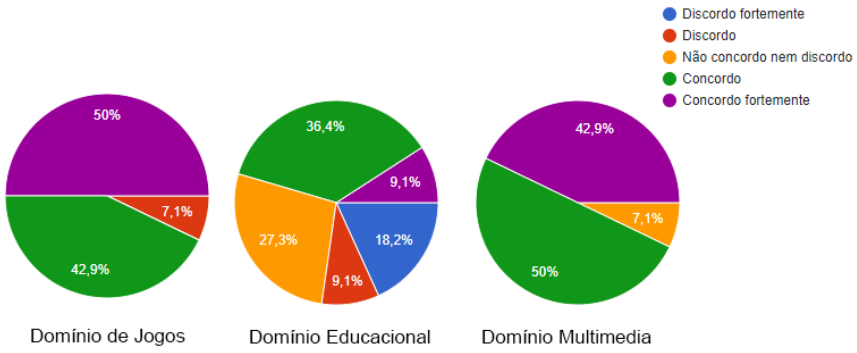


Figura 4.6: Considero que este tipo de linguagens pode ser aplicado num ambiente profissional/empresarial.

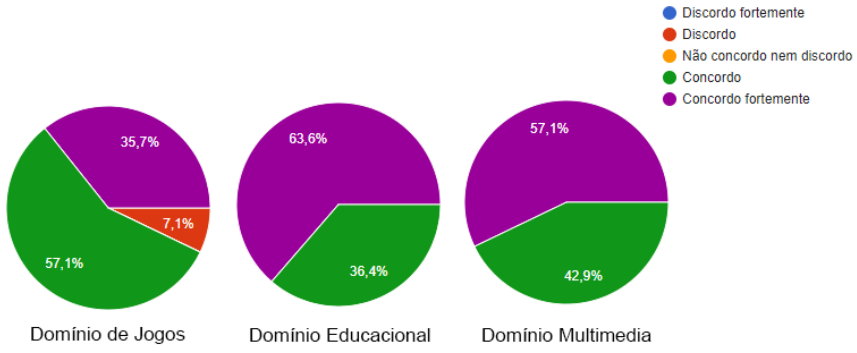


Figura 4.7: Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico.

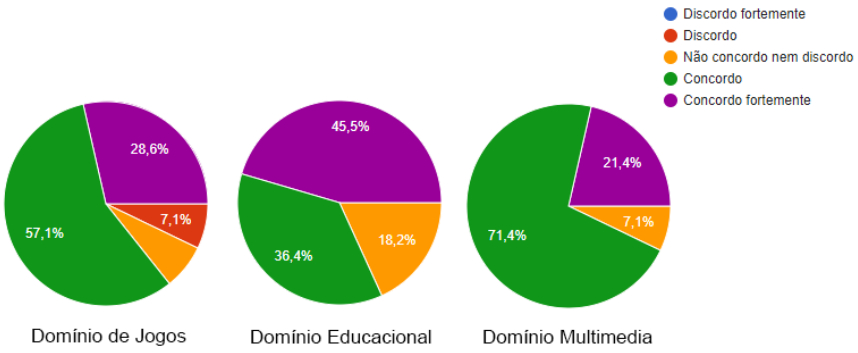


Figura 4.8: Considero que este tipo de linguagens permite uma fácil interpretação, de modo visual, do workflow do programa, tanto para o utilizador como para terceiros.

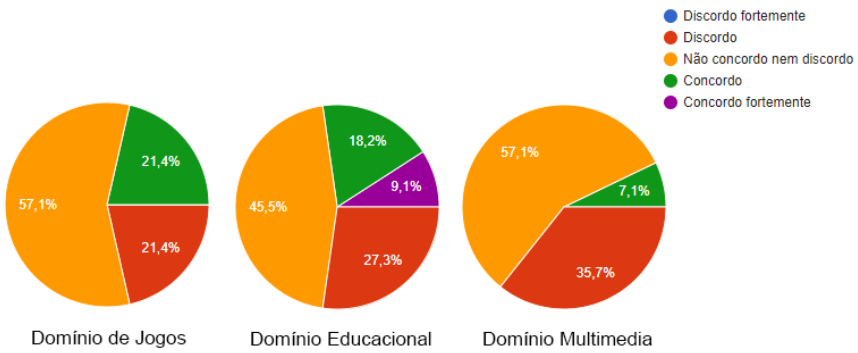


Figura 4.9: Considero que este tipo de linguagens permite detetar facilmente erros/bugs/inconsistências no programa.

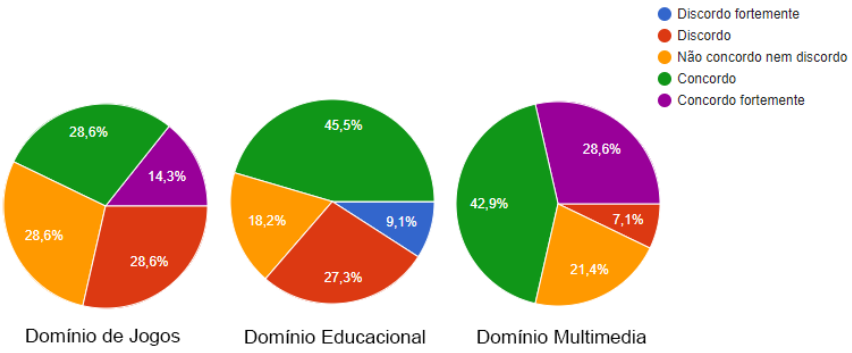


Figura 4.10: Considero que com este tipo de linguagens é possível ter um leque de funcionalidades equiparável a uma linguagem de programação convencional.

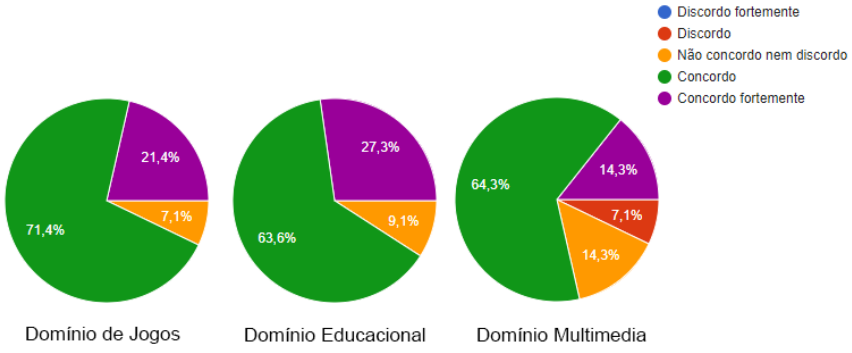


Figura 4.11: Considero que este tipo de linguagens é bastante intuitivo.

4.2.2 Domínio Multimédia

Assim como na secção anterior, ao longo desta secção estão expostos os resultados dos inquéritos relativos às perguntas do domínio Multimédia. Da mesma forma, será representada uma tabela síntese com cada pergunta deste domínio com as respetivas respostas, média e desvio padrão, depois será feita a análise a esta tabela síntese no contexto estudado e por fim serão mostrados os gráficos de cada pergunta do inquérito do domínio Multimédia.

	1	2	3	4	5	Média	Desvio Padrão	Coeficiente de Variação
Considero que a criação de prototipos com este tipo de linguagens é mais rápida do que recorrendo às linguagens de programação convencionais.	0	2	2	6	4	3,86	0,99	25,66%
Considero que com este tipo de linguagens, utilizadores em background em programção mas sim com background em arte ou musica, conseguem facilmente criar os seus trabalhos	0	1	0	8	5	4,21	0,77	18,33%
Considero que recorrendo a estas linguagens de programação visuais, os utilizadores criam os seus programas sem se preocuparem com a sua sintaxe, concentrando-se apenas na lógica e no fluxo de dados	0	2	0	6	6	4,14	0,99	23,89%
Considero que para programadores inexperientes, no que toca ao desenvolvimento deste tipo de projectos, estas linguagens fornecem suporte e documentação bastante adequados e completos	0	0	2	11	1	3,93	0,81	20,65%
considero que, sendo este tipo de linguagens do tipo dataflow, é fácil para o utilizador perceber a logica do seu programa bem como o seu fluxo	0	0	1	2	11	4,71	0,59	12,49%
Considero que os programas elaborados com este tipo de linguagens são de fácil edição, mesmo tendo uma estrutura bastantante complexa	1	1	5	6	1	3,36	1,03	30,76%
Considero que um utilizador que nunca tinha trabalhado com este tipo de linguagens consegue facilmente adaptar-se e perceber o que pode construir com as mesmas.	0	1	7	6	0	3,36	0,61	18,18%
Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.	0	4	3	6	1	3,29	0,96	29,17%
Considero que com este tipo de linguagens é fácil para um utilizador perceber os programas não elaborados pelos mesmos	0	3	5	6	0	3,21	0,77	24,04%
Por vezes a utilização destas linguagens deixa-me frustrado.	0	4	7	2	1	3,00	0,85	38,17%
Gosto de programar com este tipo de linguagens de programação	0	2	1	10	1	3,71	0,80	21,41%

Tabela 4.2: Tabela síntese de resultados do inquérito relativo às perguntas do domínio de Multimédia

Analisando agora a tabela síntese referente ao domínio de multimédia, é possível concluir que maioritariamente as respostas foram bastante positivas no que diz respeito ao uso deste tipo de linguagens neste domínio. É de realçar algumas afirmações. Como referido no capítulo de estudo deste tipo de linguagens, no que diz respeito ao domínio de multimédia as linguagens são do tipo dataflow, o que para a maioria dos utilizadores inquiridos é uma vantagem usá-las uma vez que se torna mais fácil perceber a lógica do programa gerado assim como todo o seu fluxo. Podendo muitos dos utilizadores não ter qualquer base de programação, apenas background em arte e música, pode este tipo de linguagem ser uma mais valia para os mesmos, pois para os inquiridos, com este tipo de linguagens é mais fácil criarem os seus trabalhos, conclusão esta que vem do valor da média de resposta à afirmação, que foi de 4,21, que corresponde à resposta "Concordo". A esta conclusão junta-se também o facto de a média de resposta à afirmação "Considero que recorrendo a estas linguagens de programação visuais, os utilizadores criam os seus programas sem se preocuparem com a sua sintaxe, concentrando-se apenas na lógica e no fluxo de dados"ter sido de 4,14, equivalente à resposta "Concordo". Observando agora os valores obtidos no cálculo do desvio padrão e do coeficiente de variação, numa análise geral é possível verificar que em 7 das 11 afirmações que compunham o inquérito houve alguma dispersão nas respostas dos utilizadores. Contudo, só apenas 3 desses 7 valores é que são extremamente elevados. Uma das afirmações que salta mais à vista face ao seu valor do coeficiente de variação é a afirmação número 6, com 30,76% de variação. Nesta afirmação, com uma média de resposta de 3,36 que corresponde à resposta "Não concordo nem discordo", a maior parte dos utilizadores concordou com a mesma, mas apesar disso, e como a amostra deste domínio é bastante pequena, o facto de 3 pessoas terem respondido de maneira diferente fez com que o valor deste coeficiente fosse bastante alto e por isso a conclusão retirada desta afirmação não pode ser generalizada. Por outro lado, com um valor bastante baixo de coeficiente de variação, 12,49% está a afirmação número 5. Como as opiniões dos utilizadores não divergiram quase nada umas das outras e como o valor da média de resposta desta afirmação foi de 4,71, equivalente à resposta "Concordo Fortemente", é possível concluir que as pessoas consideram que o facto deste tipo de linguagens ser dataflow, ajuda muito os utilizadores a perceberem a lógica e o fluxo dos seus programas. Relativamente à afirmação número 2, com coeficiente de variação de 18,33% pode-se também afirmar que os utilizadores inquiridos andaram todos maioritariamente à volta da mesma resposta, conclusão esta que se pode retirar devido ao valor do coeficiente de variação. Então, pode-se concluir que para estes utilizadores este tipo de linguagens é ideal para utilizadores sem background em programação mas sim em arte e música, pois como foi mostrado no capítulo de estudo destas linguagens, o tipo de programação é completamente diferente das linguagens de programação convencionais e utilizadores já com experiência na área afirmam o mesmo. Analisando agora as perguntas relativas às sensações que este tipo de linguagens transmite ao utilizador, podemos observar que para a décima afirmação, apesar de a média de resposta ter sido de 3,00, o valor do coeficiente de variação foi de 28,17% , valor este bastante elevado que traz como conclusão que a opinião dos utilizadores foi bastante heterogénea e por isso a sensação de frustração varia de utilizador para utilizador. Por outro lado, face à última questão, o valor do coeficiente de variação foi bastante mais inferior, 21,41%. Tendo

esta afirmação uma média de resposta de 3,71 e um valor baixo de variação é possível concluir que muitos utilizadores gostam de programar com este tipo de linguagens.

Como referido no início desta secção, serão agora representados os gráficos relativos a cada pergunta do inquérito deste domínio. Com exceção do primeiro gráfico, que mostra o conhecimento dos utilizadores inquiridos face a algumas linguagens de programação visual, os restantes gráficos encontram-se em percentagem.

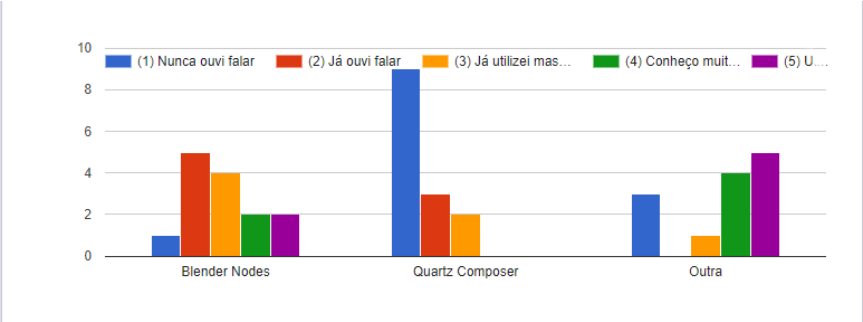


Figura 4.12: Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

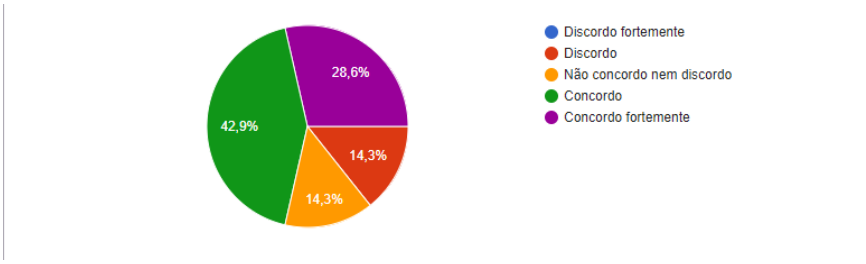


Figura 4.13: Considero que a criação de protótipos com este tipo de linguagens é mais rápida do que recorrendo às linguagens de programação convencionais.

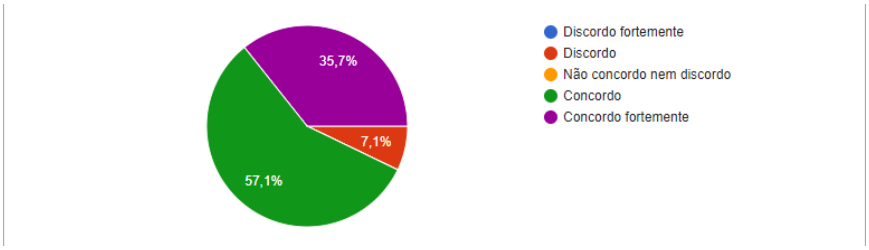


Figura 4.14: Considero que com este tipo de linguagens, utilizadores sem background em programação mas sim com background em arte ou música, conseguem facilmente criar os seus trabalhos.

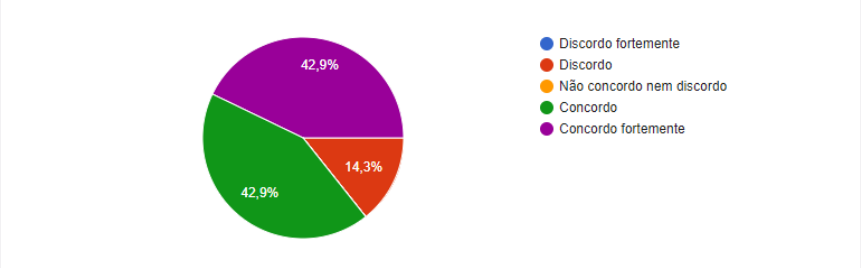


Figura 4.15: Considero que recorrendo a estas linguagens de programação visuais, os utilizadores criam os seus programas sem se preocuparem com a sua sintaxe, concentrando-se apenas na lógica e no fluxo de dados.

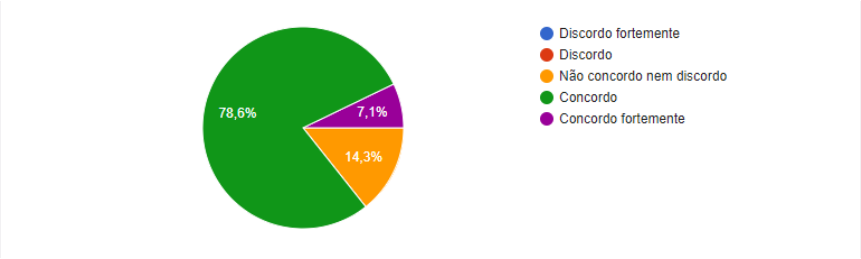


Figura 4.16: Considero que para programadores inexperientes, no que toca ao desenvolvimento deste tipo de projetos, estas linguagens fornecem suporte e documentação bastante adequados e completos.

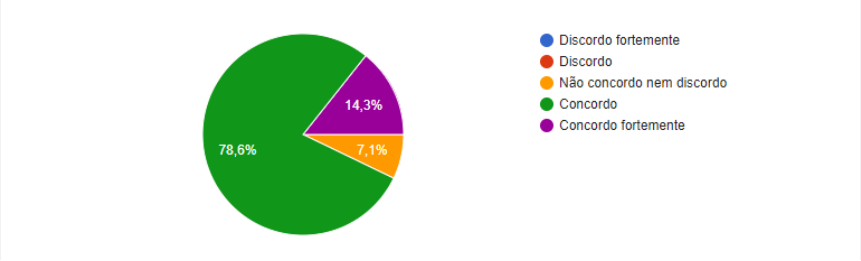


Figura 4.17: Considero que, sendo este tipo de linguagens do tipo dataflow, é fácil para o utilizador perceber a lógica do seu programa bem como o seu fluxo.

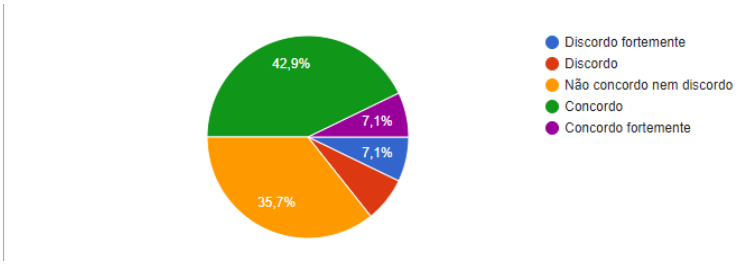


Figura 4.18: Considero que os programas elaborados com este tipo de linguagens são de fácil edição, mesmo tendo uma estrutura bastante complexa.

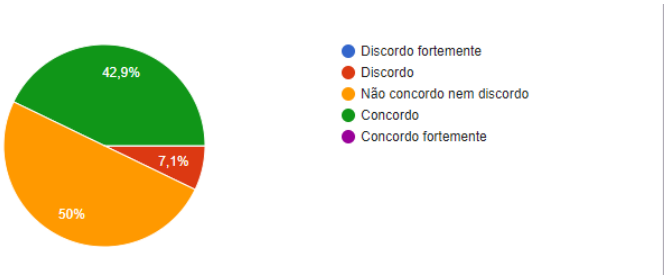


Figura 4.19: Considero que um utilizador que nunca tenha trabalhado com este tipo de linguagens consegue facilmente adaptar-se e perceber o que pode construir com as mesmas.

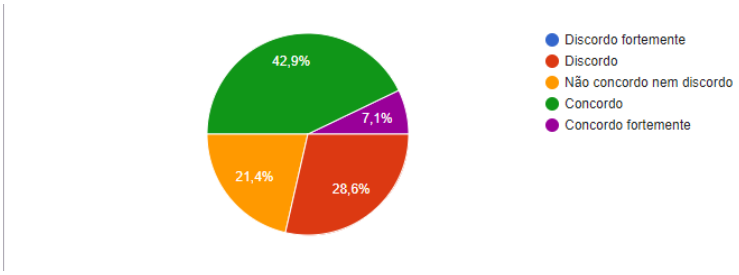


Figura 4.20: Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.

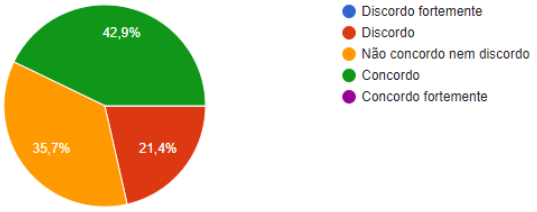


Figura 4.21: Considero que com este tipo de linguagens é fácil para um utilizador perceber o programas não elaborados pelos mesmos.

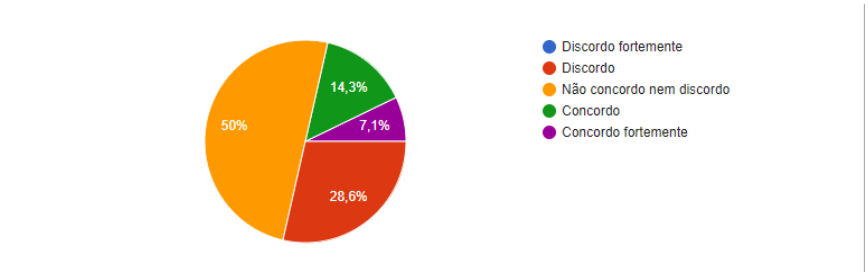


Figura 4.22: Por vezes a utilização destas linguagens deixa-me frustrado.

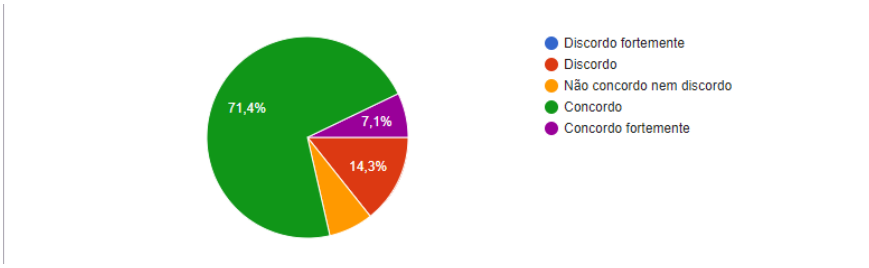


Figura 4.23: Gosto de programar com este tipo de linguagens de programação.

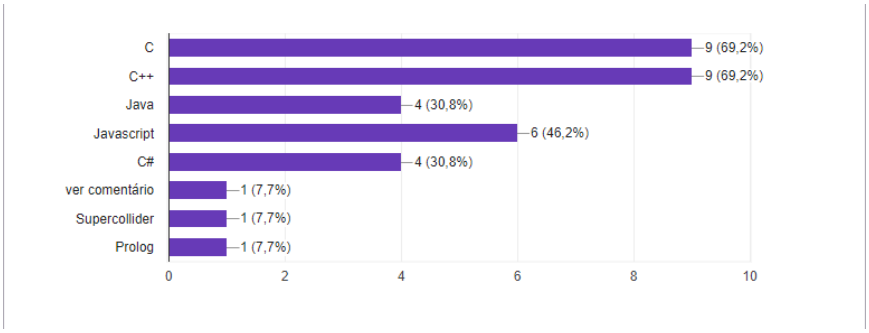


Figura 4.24: No âmbito do domínio de multimédia prefiro utilizar as linguagens de programação visuais à utilização das linguagens convencionais como:

4.2.3 Domínio Jogos

Nesta secção estão apresentados os resultados dos inquéritos relativos às perguntas do domínio de Jogos. Como nos domínios anteriores, será apresentada uma tabela síntese com cada pergunta deste domínio com as respetivas respostas, média e desvio padrão. Depois disto é feita uma análise à tabela síntese no contexto e domínio estudado e por fim serão mostrados os gráficos de cada pergunta do inquérito do domínio de Jogos.

	1	2	3	4	5	Média	Desvio Padrão	Coeficiente de Variação
Considero que tendo este tipo de linguagens o objetivo de criação de jogos, é aconselhável ser usado por utilizadores iniciantes, com poucas bases de programação, mas com curiosidade nesta área de desenvolvimento	1	0	1	7	5	4,07	1,03	25,36%
Considero que tendo este tipo de linguagens o objetivo de criação de jogos , é aconselhável ser usado por utilizadores com bastante experiência nesta área de programação.	1	4	3	5	1	3,07	1,10	35,80%
Considero que com este tipo de linguagens conseguem-se fazer jogos mais complexos do que com linguagens de programação convencionais.	5	5	0	3	0	1,93	1,12	52,50%
Considero que este tipo de linguagens permite testar os jogos enquanto o utilizador ainda está a desenvolver os mesmos.	0	1	2	6	5	4,07	0,88	21,70%
Considero que comparativamente a outras linguagens em que é possível desenvolver jogos, o desenvolvimento com recurso a estas linguagens é mais rápido e simples, uma vez que são idealizadas para este mesmo propósito.	0	0	3	5	6	4,21	0,77	18,33%
Considero que para programadores inexperientes, no que toca ao desenvolvimento de jogos, estas linguagens fornecem suporte e documentação mais adequados.	0	1	2	9	2	3,86	0,74	19,25%
Considero que as componentes disponibilizadas por este tipo de linguagens são auto explicativos e não causam dúvidas ao utilizador relativamente aos dados e tipos de dados que deve introduzir	0	3	2	9	0	3,43	0,82	23,94%
Considero que uma das grandes vantagens deste tipo de linguagens é a fácil geração de protótipos.	0	0	0	7	7	4,50	0,50	11,11%
Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.	0	3	7	4	0	3,07	0,70	22,90%
Considero que com este tipo de linguagens é fácil para um utilizador perceber os programas não elaborados pelos mesmos.	0	0	9	5	0	3,36	0,48	14,27%
Por vezes a utilização destas linguagens deixa-me frustrado.	1	3	8	2	0	2,79	0,77	27,74%
Gosto de programar com este tipo de linguagens de programação.	0	1	2	11	0	3,71	0,59	15,86%

Tabela 4.3: Tabela síntese de resultados do inquérito relativo às perguntas do domínio de Jogos

Após uma reflexão sobre os resultados obtidos nas respostas do inquérito do domínio de jogos, é possível observar alguns valores que saltam à vista, tanto pela positiva como pela negativa. Começando pela afirmação "Considero que com este tipo de linguagens conseguem-se fazer jogos mais complexos do que com linguagens de programação convencionais.", é possível perceber pelas respostas dos utilizadores inquiridos que, para os mesmos, as linguagens de programação convencionais são bastante mais poderosas do que as linguagens de programação visuais no que diz respeito à criação de jogos, uma vez que a grande maioria dos utilizadores concordou com a afirmação, tendo 5 dos mesmo respondido com total concordância. Por outro lado, para a maior parte dos utilizadores inquiridos, este tipo de linguagens torna mais rápido e simples o desenvolvimento de jogos comparativamente com outras linguagens, uma vez que foram idealizadas para este mesmo propósito e, apesar de algumas delas serem bastante simples, também há outras linguagens visuais complexas que permitem o desenvolvimento de produtos que com uma linguagem de programação convencional não seria possível. É possível também concluir que para todos os utilizadores uma grande vantagem do uso das mesmas é a fácil geração de protótipos, 100 % dos utilizadores concordaram. Analisando agora as afirmações relativas às sensações que estas linguagens causam no utilizador que as está a usar, é possível perceber que a grande maioria gosta de programar com este tipo de linguagens, tendo a resposta a esta afirmação uma média de 3,71, que corresponde à resposta "Concordo". Apesar deste último ponto, há também bastantes utilizadores que sentem frustração quando utilizam este tipo de linguagens, tendo sido a média de resposta de 2,79. Uma das razões que poderá explicar esta última questão é o facto de para a grande maioria dos utilizadores inquiridos, com este tipo de linguagens não conseguem fazer jogos tão complexos como com outras linguagens. Contudo, o facto de os utilizadores conseguirem testar os seus jogos aquando do seu desenvolvimento e o facto de apresentarem suporte e documentação essencial pode ajudar bastante no desenvolvimento dos produtos. Passando agora para a observação dos valores do desvio padrão e coeficiente de variação é possível verificar que em 7 de 12 perguntas a opinião dos utilizadores divergiu. Contudo há alguns valores que saltam mais à vista que outros e que são mais relevantes que outros. Olhando para a afirmação número 3 "Considero que com este tipo de linguagens conseguem-se fazer jogos mais complexos do que com linguagens de programação convencionais", observa-se que o seu valor de coeficiente de variação é muito elevado, 52,5%. Nesta afirmação a média de resposta tinha sido de 2,14 que corresponde à resposta "Discordo". Agora, com este valor de coeficiente de variação observa-se que os utilizadores tiveram opiniões diferentes. Um dos fatores poderá ter sido pelo diferente nível de conhecimento deste tipo de linguagens, pois apesar de todos os inquiridos terem alguma experiência na área, uns podem ter mais que outros e por isso as opiniões divergirem tanto face a esta questão. Por outro lado, um valor bastante baixo de coeficiente de variação foi das afirmações 8 e 10, 11,11 e 14,27 respetivamente. Face à primeira afirmação é então agora possível concluir que efetivamente este tipo de linguagens permite a fácil geração de protótipos, sendo esta uma das grandes vantagens, pois a sua média de resposta foi de 4,50, como já referido anteriormente e o coeficiente de variação de 11,11%, valor bastante baixo que significa que a opinião dos utilizadores foi semelhante. Face à outra afirmação, "Considero que este tipo de linguagens é fácil para um utilizador perceber os programas não ela-

borados pelos mesmos", apesar de ter um valor de coeficiente de variação bastante baixo, com o valor da média não é possível retirar qualquer conclusão isto porque a opinião dos utilizadores foi homogénea e foi o equivalente a "Não concordo nem discordo".

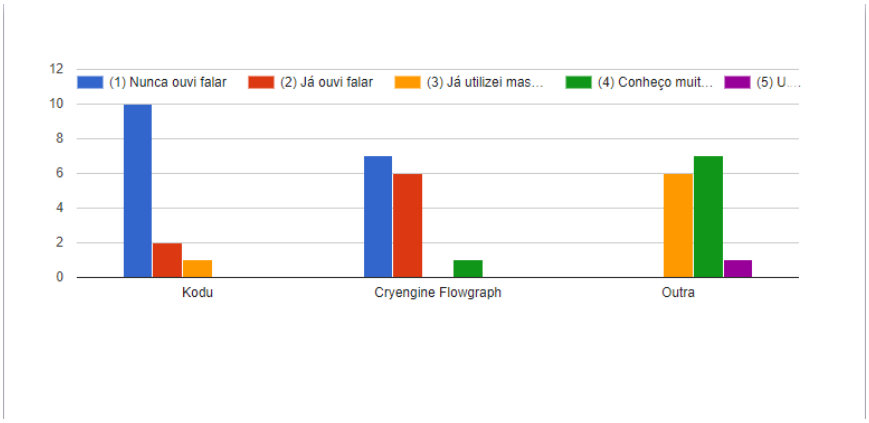


Figura 4.25: Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

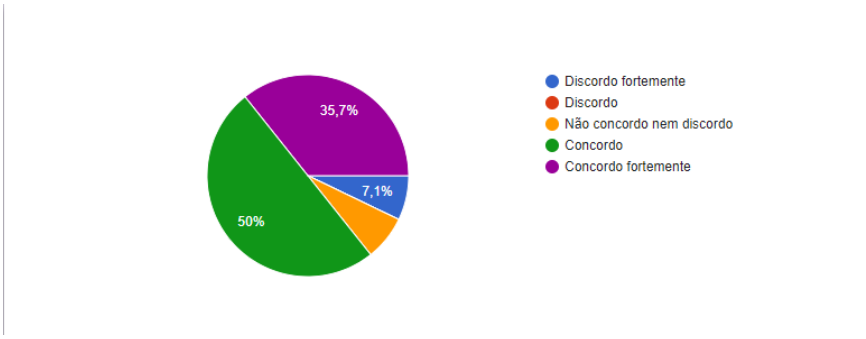


Figura 4.26: Considero que tendo este tipo de linguagens o objectivo de criação de jogos, é aconselhável ser usado por utilizadores iniciantes, com poucas bases de programação, mas com curiosidade nesta área de desenvolvimento.

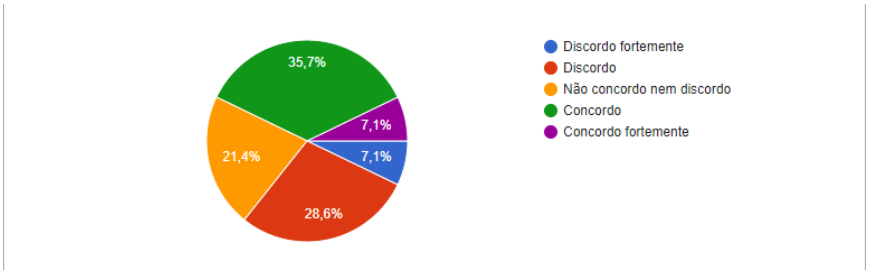


Figura 4.27: Considero que tendo este tipo de linguagens o objectivo de criação de jogos, é aconselhável ser usado por utilizadores com bastante experiência nesta área de programação.

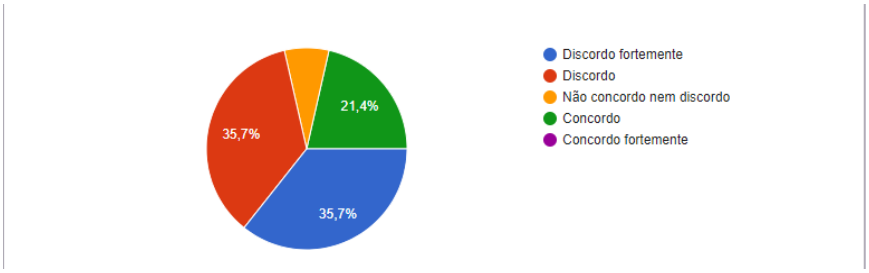


Figura 4.28: Considero que com este tipo de linguagens conseguem-se fazer jogos mais complexos do que com linguagens de programação convencionais.

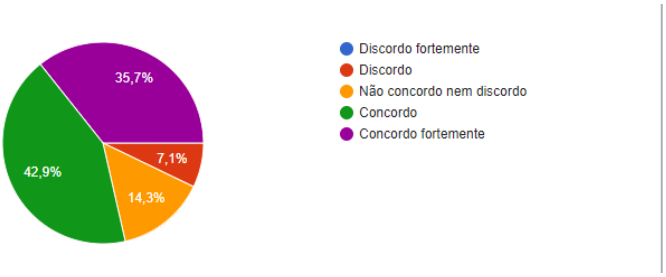


Figura 4.29: Considero que este tipo de linguagens permite testar os jogos enquanto o utilizador ainda está a desenvolver os mesmos.

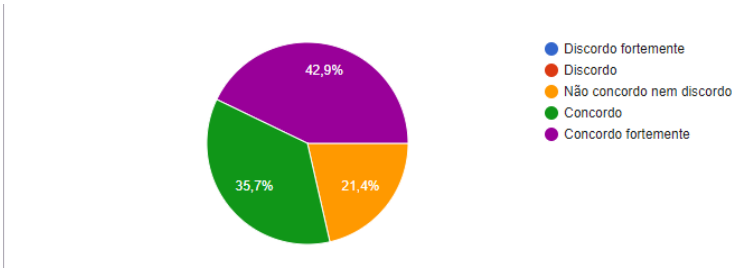


Figura 4.30: Considero que comparativamente a outras linguagens em que é possível desenvolver jogos, o desenvolvimento com recurso a estas linguagens é mais rápido e simples, uma vez que são idealizadas para este mesmo propósito.

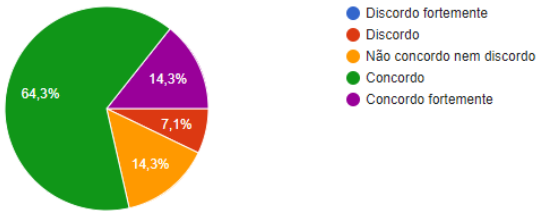


Figura 4.31: Considero que para programadores inexperientes, no que toca ao desenvolvimento de jogos, estas linguagens fornecem suporte e documentação mais adequados.

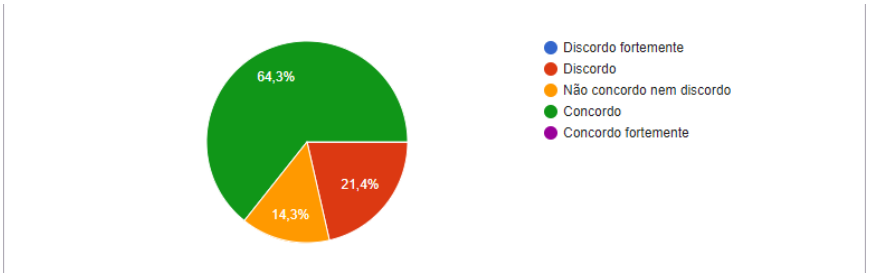


Figura 4.32: Considero que as componentes disponibilizadas por este tipo de linguagens são auto-explicativos e não causam duvidas ao utilizador relativamente aos dados e tipos de dados que deve introduzir.

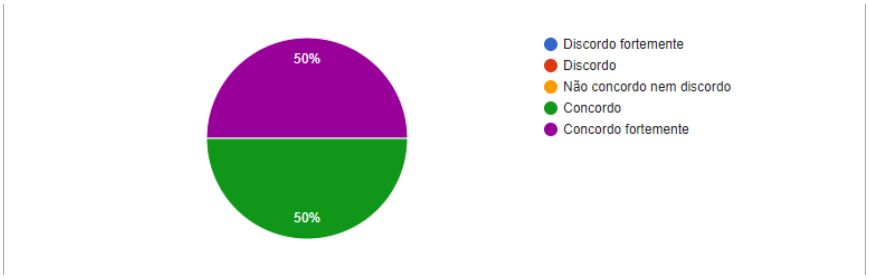


Figura 4.33: Considero que uma das grandes vantagens deste tipo de linguagens é a fácil geração de protótipos.

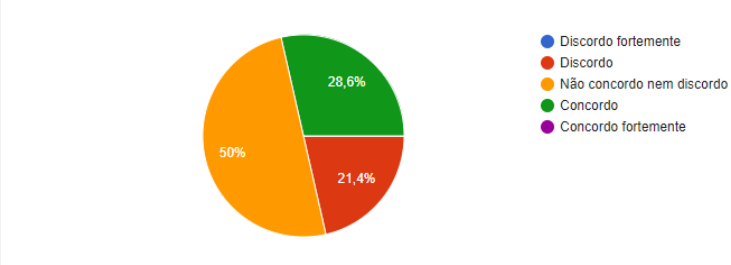


Figura 4.34: Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.

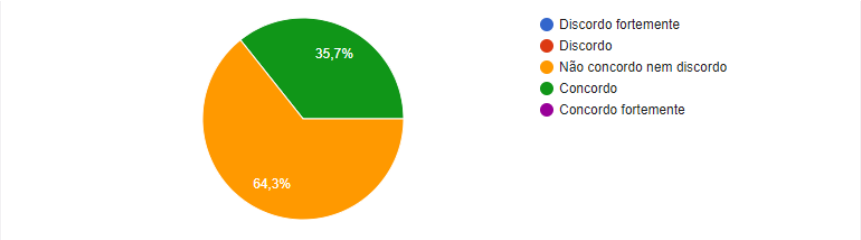


Figura 4.35: Considero que com este tipo de linguagens é fácil para um utilizador perceber o programas não elaborados pelos mesmos.

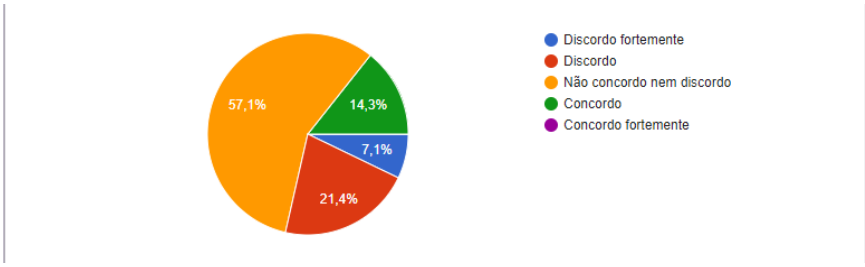


Figura 4.36: Por vezes a utilização destas linguagens deixa-me frustrado.

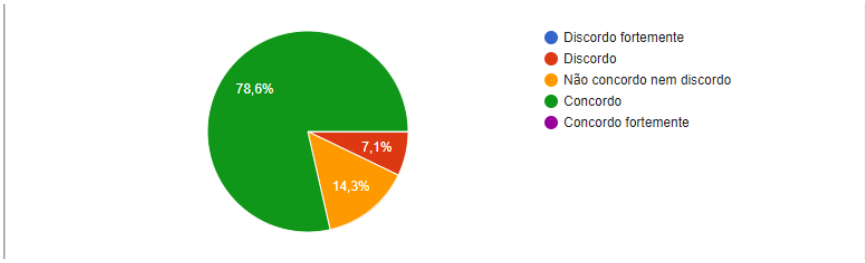


Figura 4.37: Gosto de programar com este tipo de linguagens de programação.

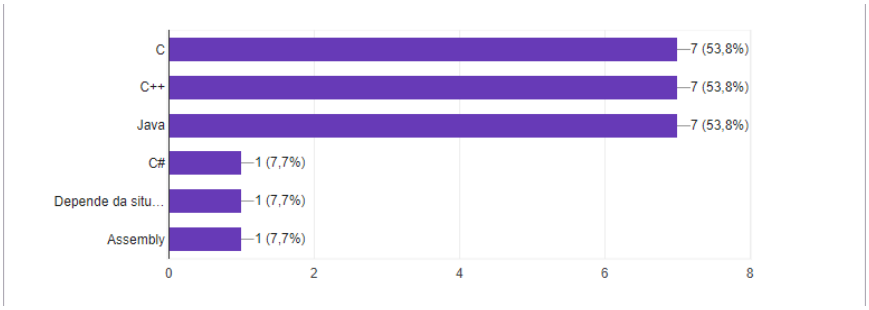


Figura 4.38: No âmbito do domínio de jogos prefiro utilizar as linguagens de programação visuais à utilização das linguagens convencionais como:

4.2.4 Domínio Educational

Nesta secção estão apresentados os resultados dos inquéritos relativos às perguntas do domínio Educacional. Como nos restantes domínios, será apresentada uma tabela síntese com cada pergunta deste domínio com as respetivas respostas, média e desvio padrão. Depois disto é feita uma análise à tabela síntese no contexto e domínio estudado e por fim serão mostrados os gráficos de cada pergunta do inquérito do domínio Educacional.

	1	2	3	4	5	Média	Desvio Padrão	Coeficiente de Variação
Considero que a usabilidade da interface destas linguagens permite uma fácil navegação no programa.	0	0	1	7	3	4,18	0,57	13,75%
Tendo este tipo de linguagens um cariz educacional, são aconselháveis a usar para se desenvolverem as capacidades base de programação de utilizadores sem noções de programação.	0	1	2	3	5	4,09	1,00	24,34%
Considero que este tipo de linguagens é de bastante fácil utilização uma vez que os seus programas são desenvolvidos praticamente pela técnica de drag and drop.	0	1	0	4	6	4,36	0,88	20,20%
Considero que este tipo de linguagens dá noções básicas de sintaxe de programação convencional uma vez que esta já está presente nas suas componentes, não sendo necessária a sua escrita.	2	0	1	6	2	3,55	1,30	36,80%
Considero que, devido ao seu modo de execução (Step-by-Step), é fácil para o utilizador visualizar o que está a acontecer no seu programa a qualquer momento da sua execução.	0	0	2	6	3	4,09	0,67	16,33%
Considero que, devido à natureza de puzzle deste tipo de linguagens, é fácil para o utilizador perceber o encadeamento do programa e quais as componentes possíveis de usar em determinadas situações.	0	0	1	6	4	4,27	0,63	14,43%
Considero que, sendo um dos objetivos deste tipo de linguagens o ensino de programação a utilizadores sem conhecimentos de programação, os avisos de erro/bugs são suficientemente explicativos para que este tipo de utilizadores, novice-programmers, percebam o que está de errado com o seu programa.	0	4	5	2	0	2,82	0,72	25,40%
Considero que uma das limitações deste tipo de linguagens é o facto de fornecerem pouca liberdade de programação, na medida em que só é possível usar as componentes predefinidas pelas plataformas.	0	2	2	3	4	3,82	1,11	29,16%
Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.	0	0	3	6	2	3,91	0,67	17,09%
Considero que com este tipo de linguagens é fácil para um utilizador perceber os programas não elaborados pelos mesmos.	0	0	2	8	1	3,91	0,51	13,16%
Por vezes a utilização destas linguagens deixa-me frustrado.	1	3	4	3	0	2,82	0,94	33,21%
Gosto de programar com este tipo de linguagens de programação.	1	2	6	2	0	2,82	0,83	29,57%

Tabela 4.4: Tabela síntese de resultados do inquérito relativo às perguntas do domínio de Educacional

Depois da observação e análise da tabela síntese do domínio Educacional, é possível concluir que os resultados obtidos foram bastante positivos. Tendo em conta o contexto de cada pergunta e respetivos valores de resposta, uma primeira conclusão a retirar é que efetivamente este tipo de linguagens pode ser uma boa abordagem no que diz respeito à utilização neste domínio. Uma das afirmações que salta mais à vista pelo seu valor de média de resposta é a seguinte: "Considero que este tipo de linguagens é de bastante fácil utilização uma vez que os seus programas são desenvolvidos praticamente pela técnica de *drag and drop*.". Nesta afirmação praticamente todos os inquiridos concordaram totalmente, tendo a média de resposta um valor de 4,36, o que significa que utilizadores já com alguma experiência em linguagens de programação visual do domínio Educacional acham que a técnica *drag and drop* é bastante simples e fácil de compreender o que faz com que a aprendizagem da linguagem seja bastante mais simples. Outra ponto que se destaca é a simplicidade das interfaces deste tipo de linguagens. Como a maior parte dos inquiridos respondeu, é fácil navegar nos programas devido à usabilidade da interface destas linguagens, o que facilita o trabalho e aprendizagem de todos os utilizadores das mesmas. Relativamente à afirmação "Tendo este tipo de linguagens um cariz educacional, são aconselháveis a usar para se desenvolverem as capacidades base de programação de utilizadores sem noções de programação.", a sua média de resposta foi de 4,09 o que permite concluir que efetivamente este tipo de linguagens é bastante indicado para aprendizagem de programação e o seu propósito deverá ser maioritariamente esse, o ensino de noções de programação a utilizadores sem bases da mesma. Para além disto, a maior parte dos inquiridos também refere que é fácil de perceber o encadeamento dos programas e que componentes usar em determinadas situações, o que é bastante relevante para utilizadores que acabam de entrar no Mundo da programação. Analisando agora os valores obtidos no cálculo do desvio padrão e coeficiente de variação, verifica-se que 6 em 12 afirmações apresentam respostas heterogéneas, pois apresentam um valor de coeficiente de variação superior a 20%. Uma afirmação onde a opinião dos utilizadores inquiridos foi bastante homogénea foi a primeira. Como já tinha sido referido, a média de resposta a esta afirmação foi de 4,18, que corresponde a "Concordo", agora com o valor do desvio padrão e do coeficiente de relação tão baixo pode-se concluir que todos os utilizadores tiveram praticamente a mesma opinião e portanto que a interface usual e simples destas linguagens permite uma navegação mais fácil ao longo dos programas. Por outro lado, a afirmação número 4, foi a afirmação onde o coeficiente de variação foi mais elevado , ou seja, onde a opinião dos utilizadores inquiridos divergiu mais. Nesta afirmação com média de resposta 3,55, houve respostas extremas, uma ccv causa será porque nem todos os utilizadores inquiridos poderão conhecer e ter experiência nas mesmas linguagens e por isso, o que numas linguagens pode ser bastante simples e com noções de sintaxe, noutras linguagens pode ser diferente. Já a natureza de puzzle deste tipo de linguagens permite para o utilizador perceber o encadeamento do programa, conclusão esta possível de retirar uma vez que as respostas dos utilizadores inquiridos foram homogéneas e o valor da média de resposta foi de 4,27, que corresponde a "Concordo". Face à afirmação número 8, apesar de a média de repostas induzir a conclusão de que efectivamente uma das limitações deste tipo de linguagens é o facto de fornecerem pouca liberdade de programação, já que se usam apenas as componentes

predefinidas pelas plataformas, o valor do desvio padrão, que é superior a 1, 1,11, assim como o valor do coeficiente de variação, 29,16 vêm demonstrar que a resposta dos utilizadores a esta afirmação não foi assim tão precisa. Apesar de mais de metade dos utilizadores concordarem com a afirnação, 4 inquiridos não foram da mesma opinião, o que, numa amostra pequena como esta, é de realçar. Como tal, uma conclusão a retirar é que dependendo daquilo que se quer elaborar e do tipo de linguagem que um utilizador usa, as limitações podem variar, assim como a liberdade que um utilizador tem pode variar dependendo da linguagem usada, não se podendo generalizar para todas as linguagens e todas as suas utilizações. Observando agora as afirmações de transmissão de sensações, pode-se verificar uma grande dispersão de opniões. Para a afirmação "Por vezes a utilização destas linguagens deixa-me frustrado", pode-se observar que apesar de a média ter sido de 2,82, as opiniões foram bastante variadas, tendo contudo 4 utilizadores discordado e apenas 3 concordado. Relativamente à última afirmação, apesar do mesmo valor de média da afirmação anterior, o valor do desvio padrão e do coeficiente de variação foi menor, o que significa que mais utilizadores escolheram a mesma opção de resposta.

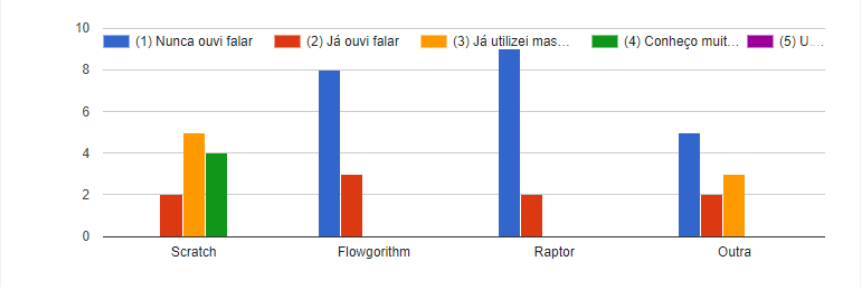


Figura 4.39: Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

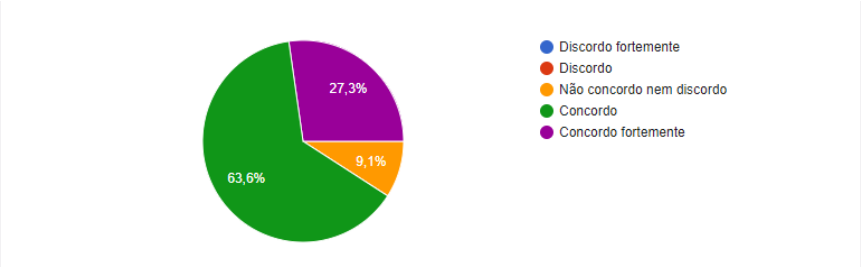


Figura 4.40: Considero que a usabilidade da interface destas linguagens permite uma fácil navegação no programa.

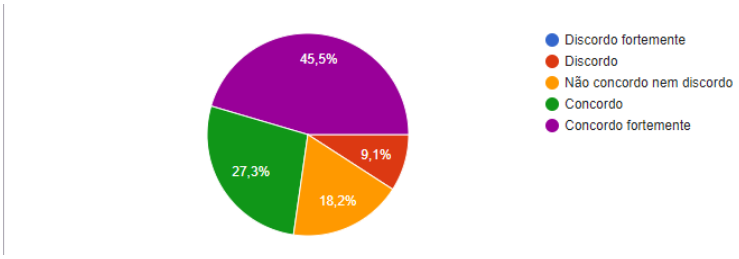


Figura 4.41: Tendo este tipo de linguagens um cariz educacional, são aconselháveis a usar para de desenvolverem as capacidades base de programação de utilizadores sem noções de programação.

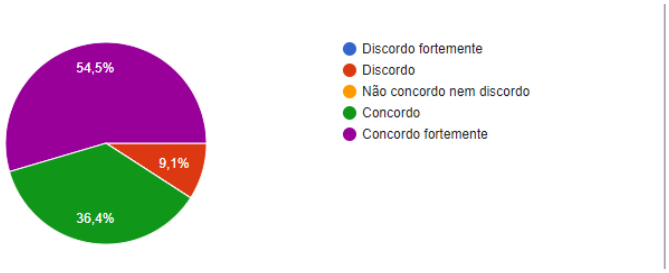


Figura 4.42: Considero que este tipo de linguagens é de bastante fácil utilização uma vez que os seus programas são desenvolvidos praticamente pela técnica de drag and drop.

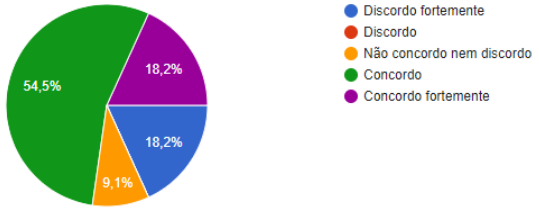


Figura 4.43: Considero que este tipo de linguagens dá noções básicas da sintaxe de programação convencional uma vez que ela está já está presente nas suas componentes, não sendo necessária a sua escrita.

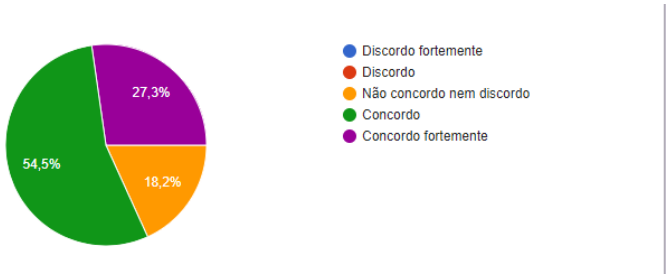


Figura 4.44: Considero que, devido ao seu modo de execução (Step-by-Step), é fácil para o utilizador visualizar o que está a acontecer no seu programa a qualquer momento da sua execução.

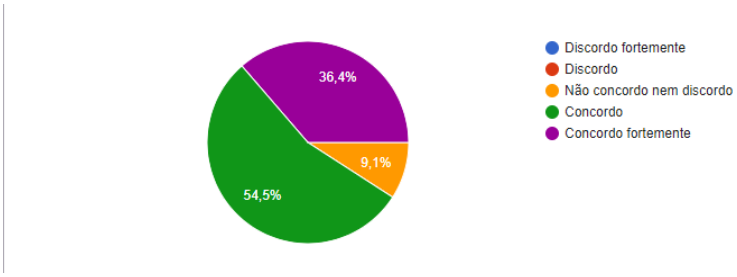


Figura 4.45: Considero que, devido à natureza de puzzle deste tipo de linguagens, é fácil para o utilizador perceber o encadeamento do programa e quais as componentes possíveis de usar em determinadas situações.

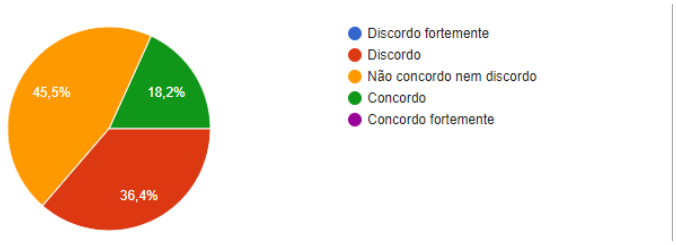


Figura 4.46: Considero que, sendo um dos objetivos deste tipo de linguagens o ensino de programação a utilizadores sem conhecimentos de programação, os avisos de erro/bugs são suficientemente explicativos para que este tipo de utilizadores, novice-programmers, percebam o que está de errado com o seu programa.

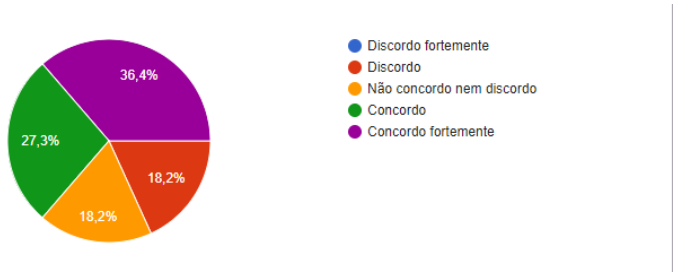


Figura 4.47: Considero que uma das limitações deste tipo de linguagens é o facto de fornecerem pouca liberdade de programação, na medida em que só é possível usar as componentes predefinidas pelas plataformas.

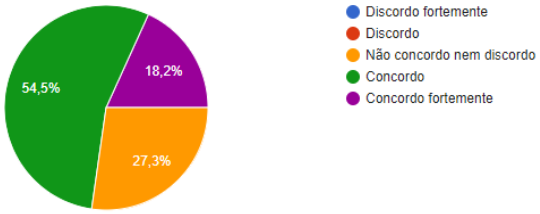


Figura 4.48: Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo.

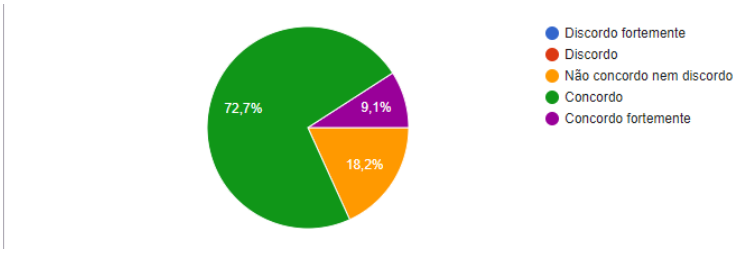


Figura 4.49: Considero que com este tipo de linguagens é fácil para um utilizador perceber o programas não elaborados pelos mesmos.

Figura 4.50: Por vezes a utilização destas linguagens deixa-me frustrado.

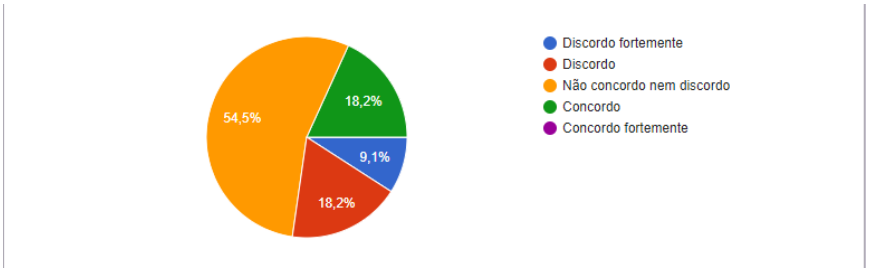


Figura 4.51: Gosto de programar com este tipo de linguagens de programação.

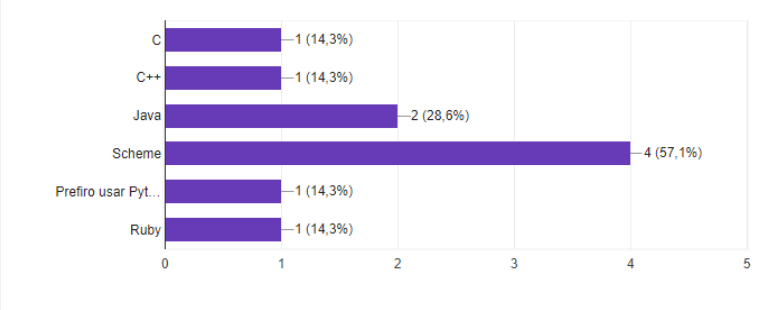


Figura 4.52: No âmbito do domínio educacional prefiro utilizar as linguagens de programação visuais à utilização das linguagens convencionais como:

4.3 Ameaças de Validação

Apesar dos resultados obtidos terem sido bastante analisados e estudados, há vários fatores que podem afetar as conclusões retiradas dos mesmos e cuja interferência não é possível de controlar. Nesta subsecção vão ser expostas algumas ameaças de validação ao meu estudo, tendo como objetivo fornecer uma correta interpretação dos resultados das afirmações, pois há alguns fatores que podem distorcer estes mesmos resultados. Uma questão que pode afetar a validade das conclusões retiradas é o número de utilizadores que compunham a amostra. Apesar, de serem utilizadores com alguma experiência no tema em estudo, sendo um grupo pequeno de inquiridos, os resultados podem não corresponder à realidade geral, pois apesar deste grupo ter experiência em cada domínio estudado, sendo um grupo pequeno não é possível generalizar para o resto dos utilizadores com experiência nesta área, que podem ter diferentes opiniões e por isso, poderiam responder de forma diferente às perguntas que compunham os inquéritos. Outro dos fatores que pode afetar as conclusões retiradas é o background dos utilizadores que responderam ao inquérito. Apesar de terem afirmado ter alguma experiência na área, é impossível comparar o grau de conhecimento entre todos os utilizadores e por isso, uma pergunta para um utilizador com muita experiência na área terá uma resposta diferente do que para um utilizador com apenas alguma experiência. Um outro fator que pode afetar os resultados e portanto distorce-los é a falta de motivação no preenchimento dos inquéritos por parte dos utilizadores inquiridos. Sendo um inquérito online e ainda com algumas perguntas longas, alguns utilizadores poderiam não ter motivação suficiente para responder a todas as questões com a maior atenção e veracidade. Outra questão que pode afetar os resultados obtidos é a credibilidade das respostas obtidas dos utilizadores. Uma vez que os inquéritos foram respondidos online, não é possível controlar o método de resposta de cada utilizador.

4.4 Resumo ou Conclusões

Ao longo deste capítulo foi feito um estudo sobre a opinião que utilizadores com experiência em linguagens de programação visuais nos domínios Educacional, Multimédia e Jogos têm relativamente à utilização deste tipo de linguagens. Foram analisados as respostas dos utilizadores

inquiridos e foram tratados os dados. O primeiro passo passou por agrupar as respostas por domínio, agrupando todas as respostas gerais e as restantes em três grupos diferentes. Depois disto, procedeu-se ao cálculo de alguns valores importantes para ajudar a retirar as devidas conclusões. Foi calculada a média de resposta de cada pergunta, o desvio padrão, assim como o coeficiente de variação. Após todos os cálculos estarem efetuados, foi construída uma tabela síntese de agregação por cada domínio com as respetivas afirmações e devidos valores. Na análise destas tabela síntese verificou-se que, uma conclusão a retirar foi que, para os utilizadores, este tipo de linguagens permite uma aprendizagem mais fácil e simples comparativamente às linguagens de programação convencionais. Como foi possível observar no capítulo de estudo destas linguagens, são muitas as Visual Programming Languages que apresentam bastante documentação de suporte e que são bastante simples e intuitivas, mesmo para numa primeira utilização. Conclui-se também que os utilizadores consideram que seria uma boa opção utilizar este tipo de linguagens em ambiente pedagógico. Sendo este tipo de linguagens de cariz visual, as pessoas consideram também que devido a este facto os utilizadores das mesmas conseguem interpretar com bastante facilidade os seus programas e percebe-los. Uma conclusão a retirar também é que para os inquiridos, a criação de protótipos com este tipo de linguagens é bastante mais rápida do que com linguagens de programação convencionais. Outra vantagem que advém da utilização destas linguagens, é que ao contrário das linguagens de programação convencionais, aqui, os utilizadores podem "esquecer"um pouco a sintaxe da linguagem e focar-se mais na lógica dos seus programas, pois a grande maioria destas linguagens já tem as componentes bem definidas e não permitem grandes alterações, o que por um lado é bastante vantajoso, pois a criação dos programas torna-se mais rápida e simples mas, por outro lado, pode limitar um pouco a liberdade de programação de cada utilizador. Isto porque algumas linguagens de programação visuais podem ser limitadas às suas componentes. Por fim, e relacionando agora todos os valores obtidos de todos os domínios é possível observar que no o domínio Educacional foi onde se verificou a maior quantidade de respostas positivas face à utilização deste tipo de linguagens. Por isso, face à opinião dos utilizadores inquiridos, é possível observar que a sua opinião é que este será o domínio onde este tipo de linguagens poderá ser mais facilmente inserido. Após a leitura de vários artigos, conclui-se também que efetivamente neste domínio há bastantes estudos à cerca da utilização deste tipo de linguagens. Quanto ao domínio onde as repostas foram consideradas mais negativas, relativamente à utilização destas linguagens, foi o domínio de Jogos. Face a todas as análises feitas, verifica-se que este tipo de linguagens pode trazer muitas vantagens e também desvantagens com a sua utilização, mas que em certos domínios e com certos utilizadores seria uma boa opção a sua utilização.

Capítulo 5

Conclusões e Trabalho Futuro

O objectivo desta dissertação era a exploração da área da *Visual Programming* a partir da perspectiva do *end-user*, com o intuito de perceber quais as vantagens e desvantagens da utilização da mesma assim como a aceitação deste tipo de linguagens por parte de utilizadores experientes nas mesmas. Para isso, foram estudadas várias *Visual Programming Languages*, agrupadas por três domínios diferentes, domínio Educacional, domínio de Multimédia e domínio de Jogos, e foi realizado um inquérito a diferentes tipos de utilizadores com experiência nestas linguagens.

Nesta dissertação, foram então apresentados vários aspectos relativos às *Visual Programming Languages*, abordando as suas estratégias, a sua classificação e alguns prós e contras já referidos por vários autores. Para a análise final e respectivas conclusões sobre a utilização destas lingaugens, foi feito um inquérito onde a opinião dos utilizadores inquiridos foi devidamente tratada e analisada.

Como referido no capítulo 2 e comprovado com o inquérito elaborado, uma das vantagens da utilização deste tipo de linguagens é que a passagem da abstração para o nível visual torna a compreensão dos programas muito mais simples do que utilizando as linguagens de programação convencionais.

É possível reconhecer também que o ambiente em que a programação visual se encontra mais aceite pelos utilizadores, e onde se encontram mais vantagens aquando da sua utilização, é no domínio Educacional. Este tipo de linguagens, construídas com o propósito Educacional, devido à sua simplicidade, torna a programação um actividade mais fácil e mais interactiva para utilizadores sem qualquer base em programação. Apesar disto, foram também apontados alguns problemas na sua utilização, como o tratamento de erros que estas linguagens apresentam e a pouca liberdade que dão aos utilizadores. Contudo, se o objectivo das mesmas é o ensino de Programação a utilizadores sem qualquer base na mesma, penso que esta limitação não se aplica e que a utilização destas linguagens no ensino seria uma boa opção para as instituições de ensino.

Quanto aos restantes domínios, as opiniões dos utilizadores divergiram um pouco. Por um lado há algumas vantagens na sua utilização, como a necessidade de o utilizador pensar na sintáxe

do programa é muito mais reduzida pois as componentes já se encontram pré-definidas. O tempo gasto na elaboração dos programas também é reduzido pois as linguagens foram construídas com o propósito de utilização no domínio específico. Sendo estas linguagens maioritariamente do tipo dataflow, a compreensão dos programas e respectiva análise também é mais fácil e simples comparativamente com linguagens de programação convencionais.

Por outro lado, depois do estudo elaborado também foi possível identificar várias restrições na utilização das *Visual Programming Languages*. A principal desvantagem passa por a complexidade dos programas ser mais reduzida relativamente às linguagens de programação convencionais. Como foi possível observar nas várias interfaces destas linguagens, o espaço que o programa ocupa na interface uma maior área do que as linguagens convencionais, o que pode causar alguma confusão aos utilizadores se os programas não estiverem limpos e bem organizados. Sendo este tipo de linguagens maioritariamente desenvolvidos para a utilização em domínios específicos, a utilização das mesmas podem-se tornar desvantajosa pois não são desenvolvidas para solucionar as necessidades e problemas gerais dos utilizadores. Assim, a reutilização para outros projectos ou problemas será limitada.

É então possível concluir que efetivamente tendo o utilizador o problema definido, a solução e o domínio bem definido, e sendo este domínio possível de enquadrar com este tipo de linguagens, acho que a utilização das *Visual Programming Languages* será uma boa abordagem. No entanto, o único domínio em que digo com toda a certeza que será bastante importante e melhorará bastante alguns problemas dos dias de hoje, será a utilização destas linguagens no domínio Educacional.

5.1 Trabalho Futuro

Uma proposta para trabalho futuro seria a continuidade do estudo das *Visual Programming Languages* e a respectiva recolha de Padrões entre as mesmas. Tendo também em consideração os diferentes domínios e os diferentes tipos de utilizadores que poderão usar estas linguagens. O objectivo seria então providenciar aos designers de *Visual Programming Languages* conhecimento específico sobre como e quando utilizar certas metáforas e representações.

Referências

[Ahm99] Rodina Ahmad. Visual languages: A new way of programming. *Malaysian Journal of Computer Science*, 12(1):76–81, 1999.

[BBB+94] Margaret M Burnett, Marla J Baker, Carisa Bohus, Paul Carlson, Pieter J Van Zee e Sherry Yang. The scaling-up problem for visual programming languages. 1994.

[BD04] M. Boshernitsan e M. Downes. Visual programming languages: A survey. *Computer*, (December), 2004. URL: http://nitsan.org/~maratb/cs263/.

[Ber07] Dane Bertram. Likert Scales... are the meaning of life :. *University of Calagary, Department of Computer Science*, page pages.cpsc.ucalgary.ca/˜saul/wiki/uploads/CPSC681/, 2007. doi:10.1002/9780470479216.corpsy0508.

[Ble] Blender. Node Editor — Blender Manual. URL: https://docs.blender.org/manual/en/dev/editors/node_editor/index.html.

[Bur99] Margaret Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 32(1-3):275–283, 1999. URL: ftp://ftp.cs.orst.edu/pub/burnett/whatIsVP.pdf, doi:10.1002/047134608X.W1707.

[Chr06] Per Christensson. "Recursion Definition."TechTerms, 2006. URL: https://techterms.com/definition/recursion.

[Coo] Devin Cook. Flowgorithm - Flowchart Programming Language. URL: http://www.flowgorithm.org/index.htm.

[Cor] Microsoft Corporation. Kodu | Home. URL: https://www.kodugamelab.com/.

[CPR04] José Carlos, Rocha Pereira e Clevi Elena Rapkiewicz. O Processo de Ensino-Aprendizagem de Fundamentos de Programação : Uma Visão Crítica da Pesquisa no Brasil. *Anais do XII Workshop sobre Educação em Computação (SBC)*, 2004.

[CRY] CRYENGINE. CRYENGINE | The complete solution for next generation game development by Crytek. URL: https://www.cryengine.com/.

[Dij89] Edsger W Dijkstra. On the cruelty of really teaching computing science. *Communications of the ACM*, 32(12):1397–1414, 1989. doi:10.1017/CBO9781107415324.004.

[GLK] MIT Media Lab Grupo Lifelong Kindergarten. Scratch - Imagine, Program, Share. URL: https://scratch.mit.edu/.

REFERÊNCIAS

[GP92] T.R.G. Green e M. Petre. When Visual Programs are Harder to Read than Textual Programs. *Human-Computer Interaction: Tasks and Organisation*, (1987):167–180, 1992. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.1633.

[GR90] Eric J. Golin e Steven P. Reiss. The specification of visual language syntax. *Journal of Visual Languages and Computing*, 1(2):141–157, 1990. doi:10.1016/S1045-926X(05)80013-8.

[Guz03] Mark Guzdial. A media computation course for non-majors. *Proceedings of the 8th annual conference on Innovation and technology in computer science education - ITiCSE '03*, page 104, 2003. URL: http://portal.acm.org/citation.cfm?doid=961511.961542, doi:10.1145/961511.961542.

[Kou16] Mathieu K Kourouma. Capabilities and Features of Raptor , Visual Logic , and Flowgorithm for Program Logic and Design Capabilities and Features of Raptor , Visual Logic , and Flowgorithm for Program Logic and Design. (October), 2016.

[Mal10] Resnick R. Rusk N. Silverman B. & Eastmond E. Maloney, J. The scratch programming language and environment. *ACM Transactions on Computing Education*, 10(4):1–15, 2010. doi:10.1145/1868358.1868363.

[Mye86] Brad A Myers. Visual programming, programming by example, and program visualization: a taxonomy. In *ACM SIGCHI Bulletin*, volume 17, pages 59–66. ACM, 1986.

[Mye90] Brad Myers. Taxonomies of Visual Programming and P rogram Visualization. *Visual Languages and Computing*, 1(1):97–123, 1990.

[NH97] N. Hari; Narayanan e Roland Hübscher. VISUAL LANGUAGE THEORY: TOWARDS A HUMAN-COMPUTER INTERACTION PERSPECTIVE. page 31, 1997.

[oESSP16] College of Engineering, Computer Science e California State University Snarky Professors. Part 1 – Your First Function. pages 1–8, 2016.

[Qua] Quartz Composer Programming Guide: Introduction to Quartz Composer Programming Guide. URL: http://mirror.informatimago.com/next/developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/.

[Rod02] Susan Rodger. Introducing Computer Science Through Animation and Virtual Worlds. *Sigcse 2002*, pages 186–190, 2002. doi:10.1145/563340.563411.

[UKR] UKRocketry. Flowgorithm Using an Array. URL: https://www.youtube.com/watch?v=ThV_hsVPHiA.

[WCHM] Terry Wilson, Martin C. Carlisle, Jeff Humphries e Jason Moore. RAPTOR - Flowchart Interpreter. URL: http://raptor.martincarlisle.com/.

Anexo A

Questionários

A.1 Visual Programming Languages - Domínio Educacional

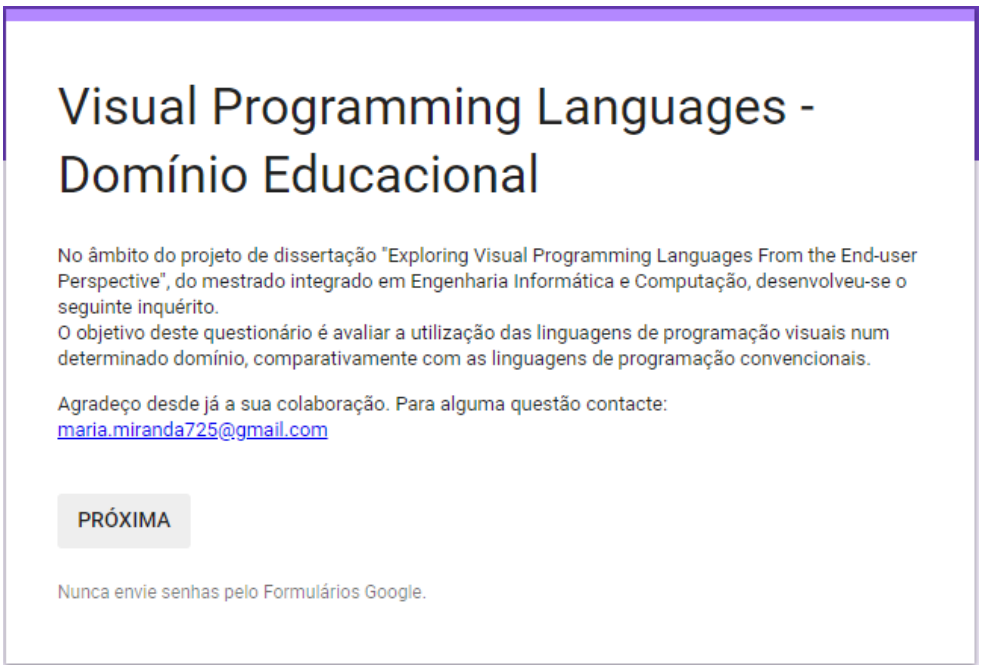
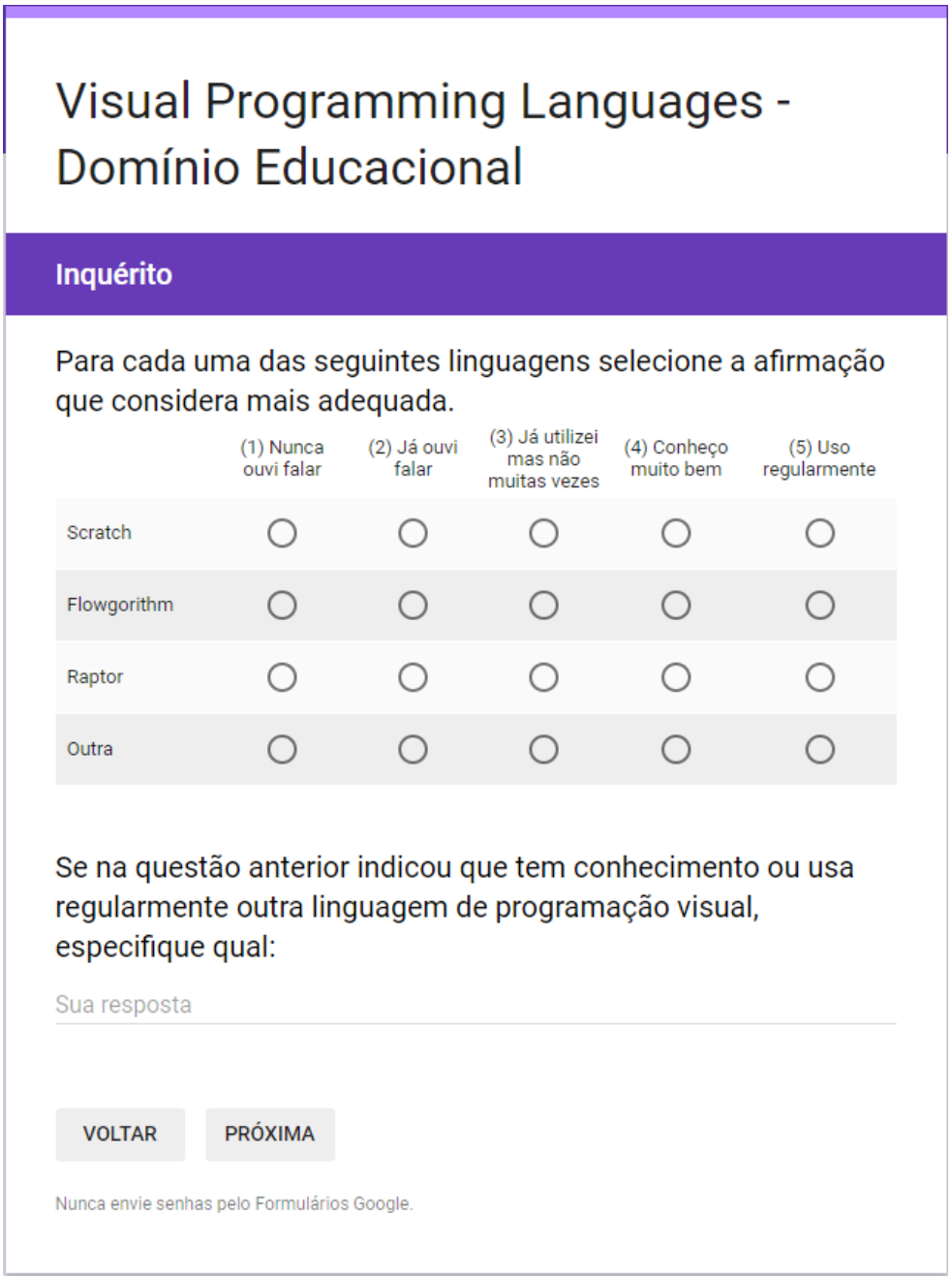


Figura A.1: Questionário Domínio Educacional



Visual Programming Languages - Domínio Educacional

Inquérito

Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

	(1) Nunca ouvi falar	(2) Já ouvi falar	(3) Já utilizei mas não muitas vezes	(4) Conheço muito bem	(5) Uso regularmente
Scratch	○	○	○	○	○
Flowgorithm	○	○	○	○	○
Raptor	○	○	○	○	○
Outra	○	○	○	○	○

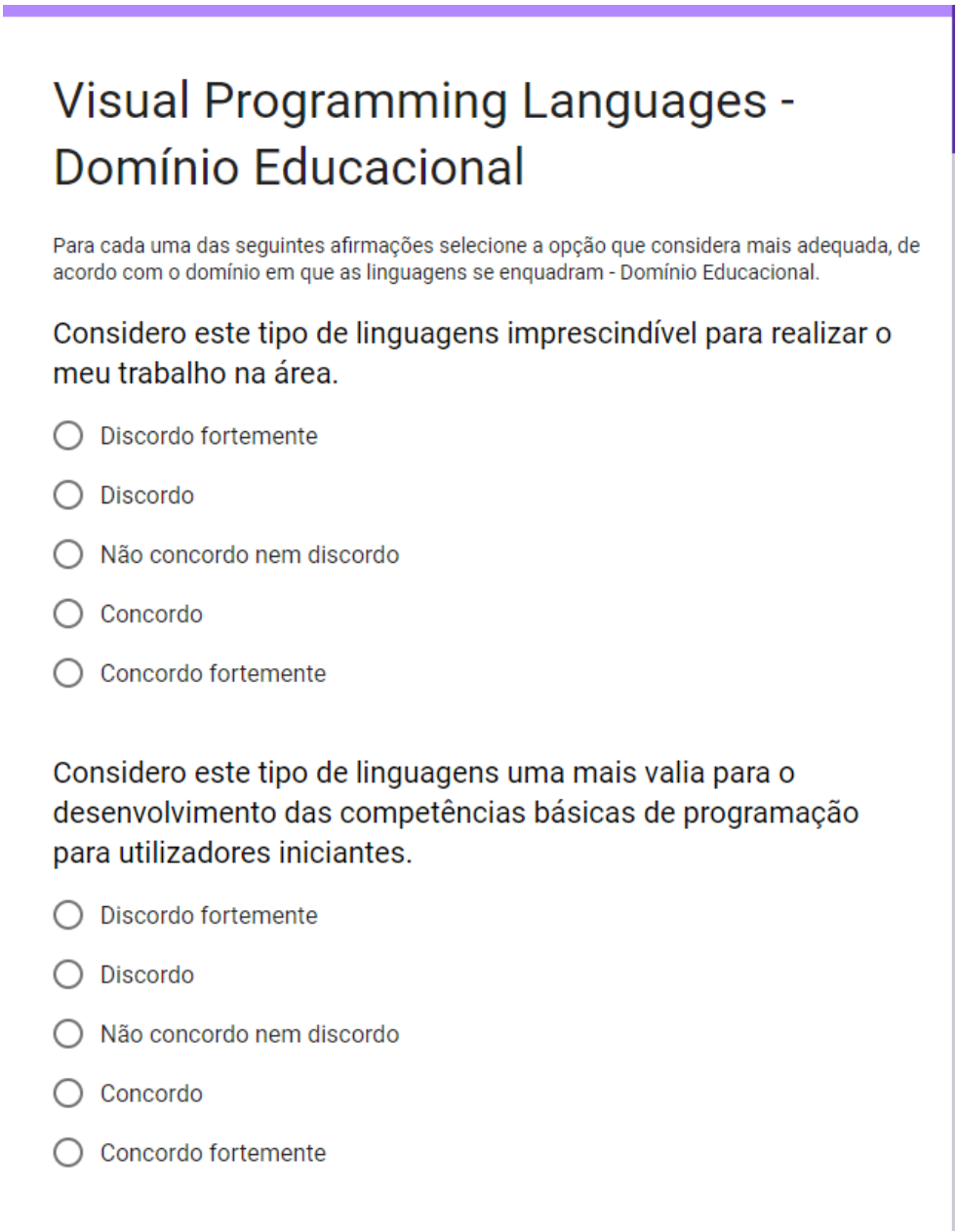
Se na questão anterior indicou que tem conhecimento ou usa regularmente outra linguagem de programação visual, especifique qual:

Sua resposta

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Figura A.2: Questionário Domínio Educacional



Visual Programming Languages - Domínio Educacional

Para cada uma das seguintes afirmações selecione a opção que considera mais adequada, de acordo com o domínio em que as linguagens se enquadram - Domínio Educacional.

Considero este tipo de linguagens imprescindível para realizar o meu trabalho na área.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.3: Questionário Domínio Educacional

Considero que este tipo de linguagens permite aos utilizadores ter uma curva de aprendizagem menor em relação às linguagens de programação convencionais.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária menor.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (18+).

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Figura A.4: Questionário Domínio Educacional

Considero que este tipo de linguagens pode ser aplicado num ambiente profissional/empresarial.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens permite uma fácil interpretação, de modo visual, do workflow do programa, tanto para o utilizador como para terceiros.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.5: Questionário Domínio Educacional

Considero que este tipo de linguagens permite detetar facilmente erros/bugs/inconsistências no programa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens é possível ter um leque de funcionalidades equiparável a uma linguagem de programação convencional.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é bastante intuitivo.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.6: Questionário Domínio Educacional

Considero que a usabilidade da interface destas linguagens permite uma fácil navegação no programa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Tendo este tipo de linguagens um cariz educacional, são aconselháveis a usar para de desenvolverem as capacidades base de programação de utilizadores sem noções de programação.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é de bastante fácil utilização uma vez que os seus programas são desenvolvidos praticamente pela técnica de drag and drop.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.7: Questionário Domínio Educacional

Considero que este tipo de linguagens dá noções básicas da sintaxe de programação convencional uma vez que ela está já está presente nas suas componentes, não sendo necessária a sua escrita.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que, devido ao seu modo de execução (Step-by-Step), é fácil para o utilizador visualizar o que está a acontecer no seu programa a qualquer momento da sua execução.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.8: Questionário Domínio Educacional

Considero que, devido à natureza de puzzle deste tipo de linguagens, é fácil para o utilizador perceber o encadeamento do programa e quais as componentes possíveis de usar em determinadas situações.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que, sendo um dos objetivos deste tipo de linguagens o ensino de programação a utilizadores sem conhecimentos de programação, os avisos de erro/bugs são suficientemente explicativos para que este tipo de utilizadores, novice-programmers, percebam o que está de errado com o seu programa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.9: Questionário Domínio Educacional

Considero que uma das limitações deste tipo de linguagens é o facto de fornecerem pouca liberdade de programação, na medida em que só é possível usar as componentes predefinidas pelas plataformas.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens é fácil para um utilizador perceber o programas não elaborados pelos mesmos.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.10: Questionário Domínio Educacional

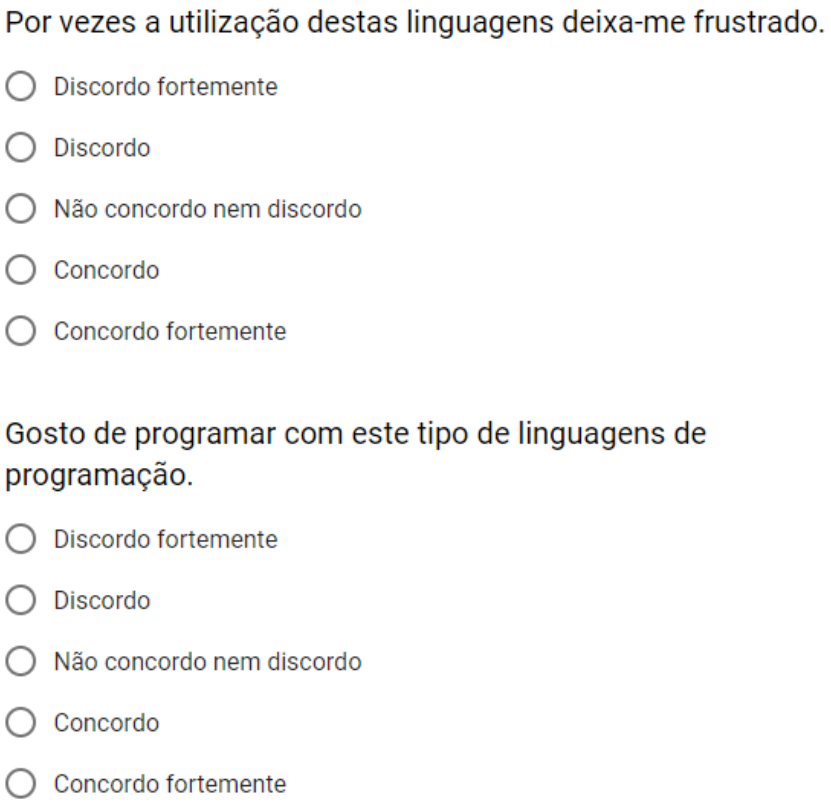


Figura A.11: Questionário Domínio Educacional

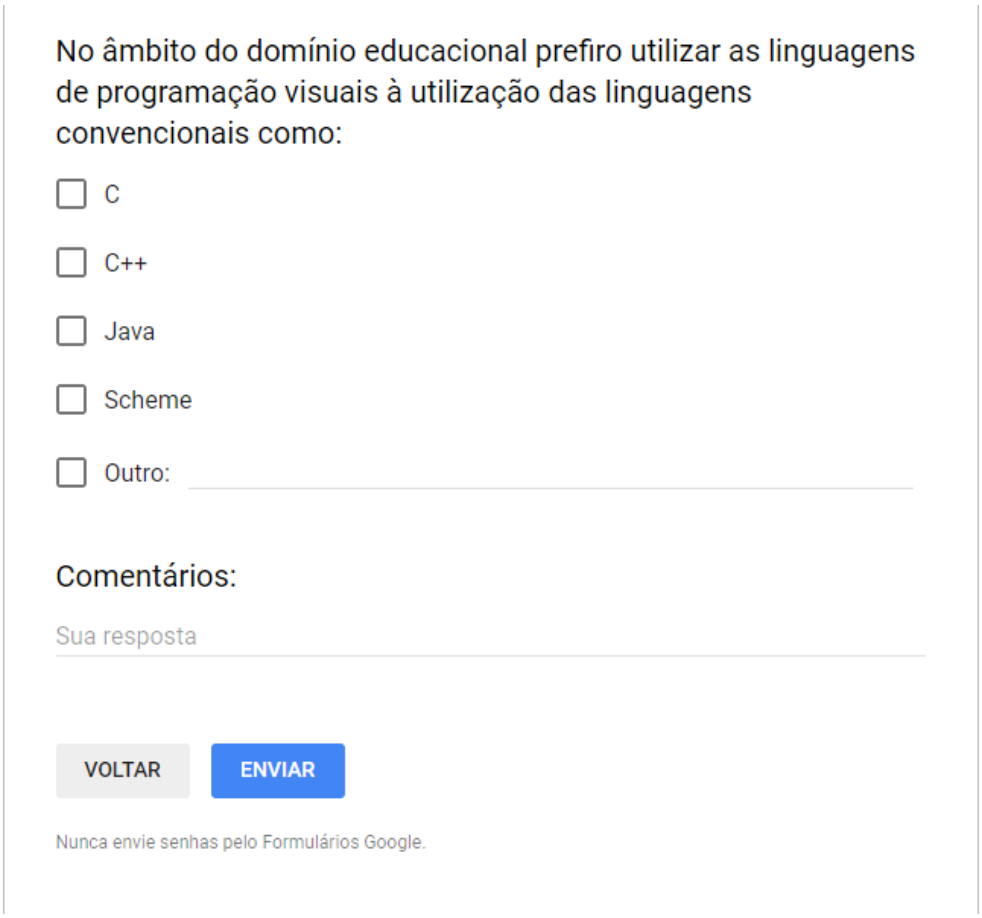
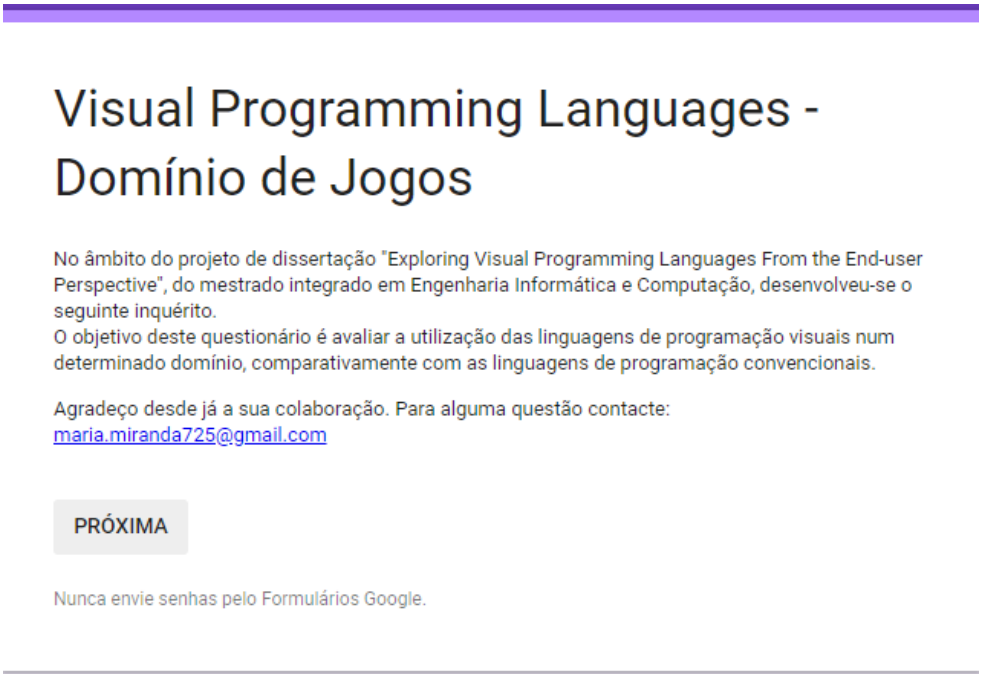


Figura A.12: Questionário Domínio Educacional

A.2 Visual Programming Languages - Domínio de Jogos



Visual Programming Languages - Domínio de Jogos

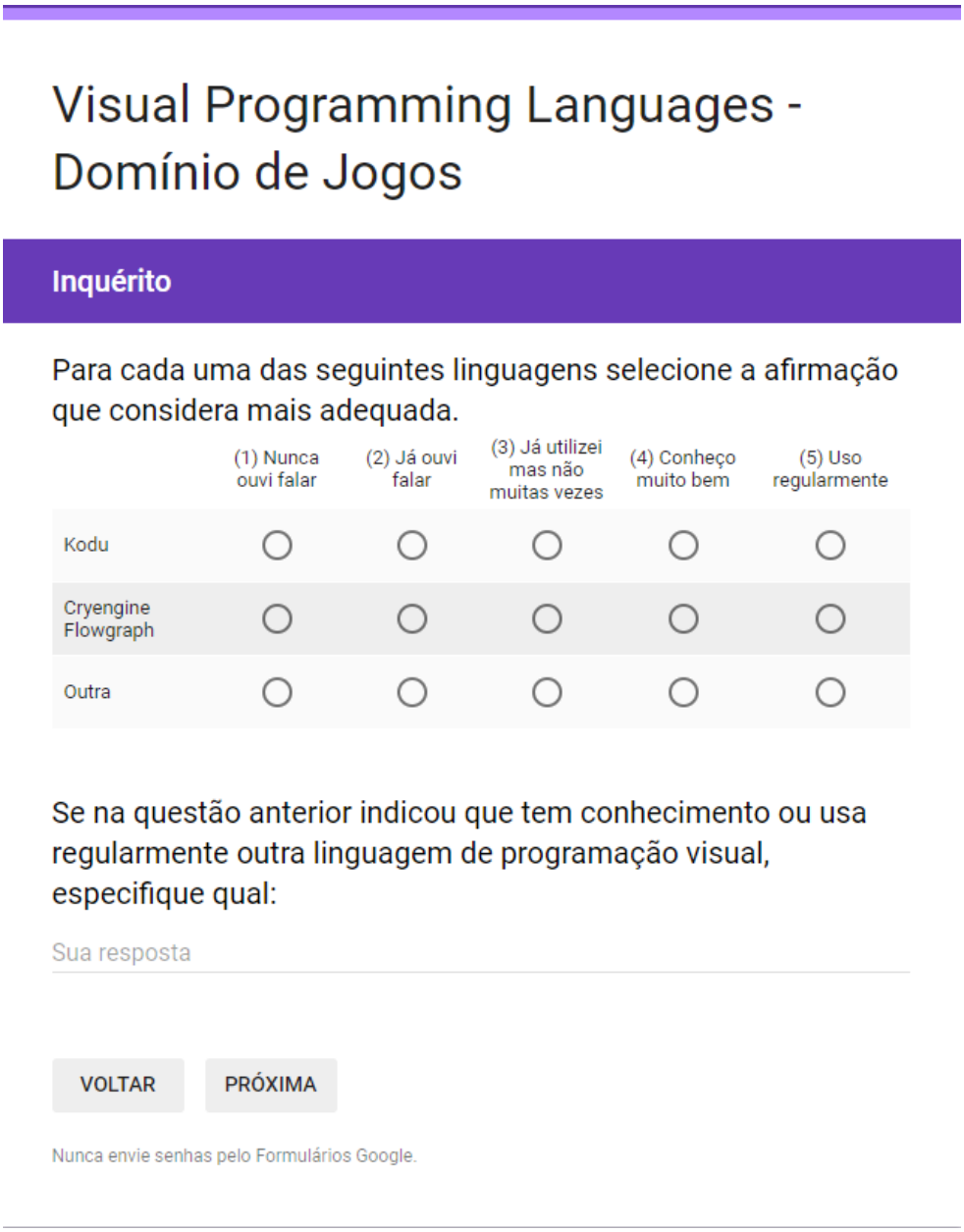
No âmbito do projeto de dissertação "Exploring Visual Programming Languages From the End-user Perspective", do mestrado integrado em Engenharia Informática e Computação, desenvolveu-se o seguinte inquérito.
O objetivo deste questionário é avaliar a utilização das linguagens de programação visuais num determinado domínio, comparativamente com as linguagens de programação convencionais.

Agradeço desde já a sua colaboração. Para alguma questão contacte:
maria.miranda725@gmail.com

PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Figura A.13: Questionário Domínio de Jogos



Visual Programming Languages - Domínio de Jogos

Inquérito

Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

	(1) Nunca ouvi falar	(2) Já ouvi falar	(3) Já utilizei mas não muitas vezes	(4) Conheço muito bem	(5) Uso regularmente
Kodu	○	○	○	○	○
Cryengine Flowgraph	○	○	○	○	○
Outra	○	○	○	○	○

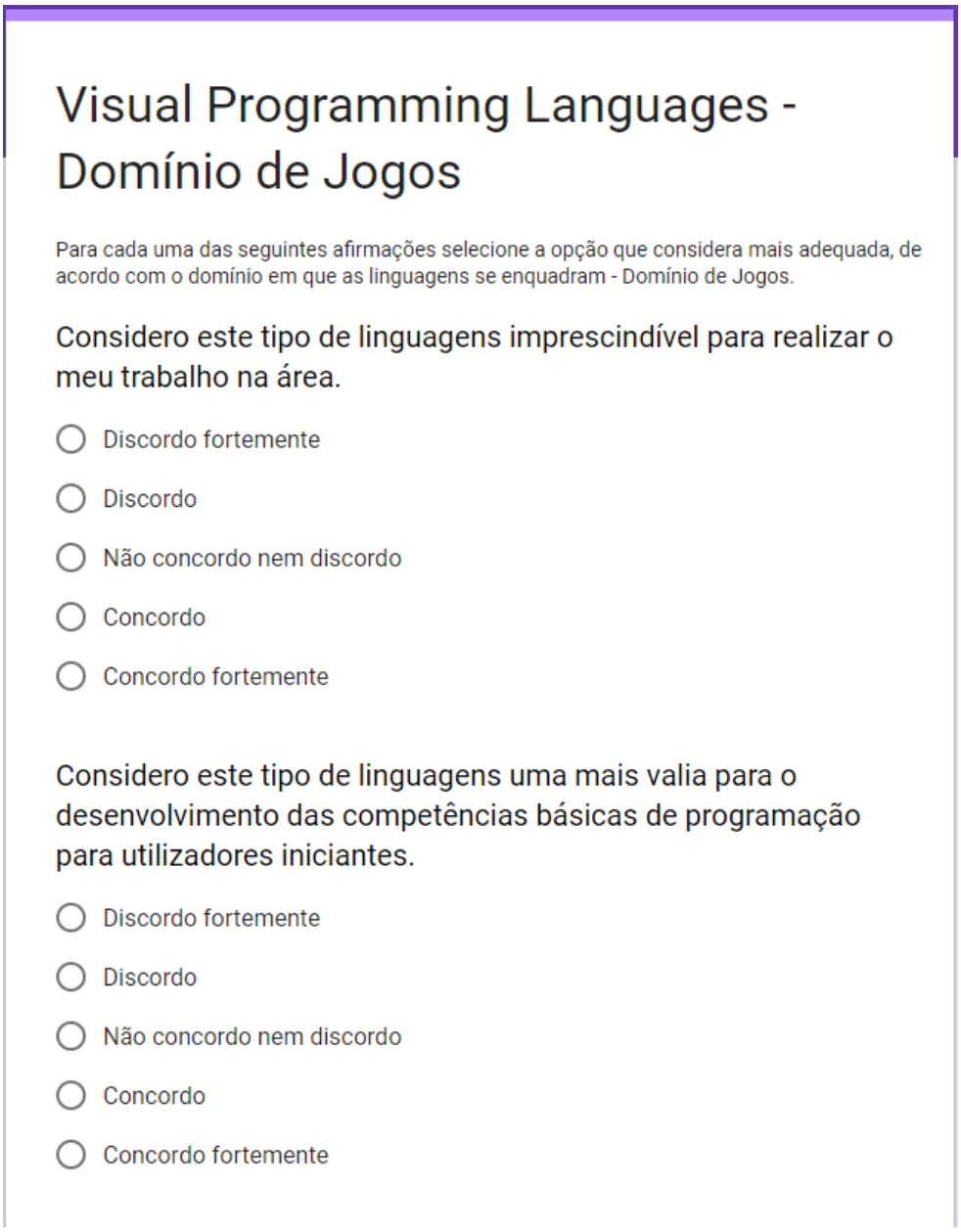
Se na questão anterior indicou que tem conhecimento ou usa regularmente outra linguagem de programação visual, especifique qual:

Sua resposta

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Figura A.14: Questionário Domínio de Jogos



Visual Programming Languages - Domínio de Jogos

Para cada uma das seguintes afirmações selecione a opção que considera mais adequada, de acordo com o domínio em que as linguagens se enquadram - Domínio de Jogos.

Considero este tipo de linguagens imprescindível para realizar o meu trabalho na área.

- Discordo fortemente
- Discordo
- Não concordo nem discordo
- Concordo
- Concordo fortemente

Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes.

- Discordo fortemente
- Discordo
- Não concordo nem discordo
- Concordo
- Concordo fortemente

Figura A.15: Questionário Domínio de Jogos

Considero que este tipo de linguagens permite aos utilizadores ter uma curva de aprendizagem menor em relação às linguagens de programação convencionais.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária menor.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (18+).

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.16: Questionário Domínio de Jogos

Considero que este tipo de linguagens pode ser aplicado num ambiente profissional/empresarial.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens pode ser aplicado num ambiente pedagógico.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens permite uma fácil interpretação, de modo visual, do workflow do programa, tanto para o utilizador como para terceiros.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.17: Questionário Domínio de Jogos

Considero que este tipo de linguagens permite detetar facilmente erros/bugs/inconsistências no programa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens é possível ter um leque de funcionalidades equiparável a uma linguagem de programação convencional.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é bastante intuitivo.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.18: Questionário Domínio de Jogos

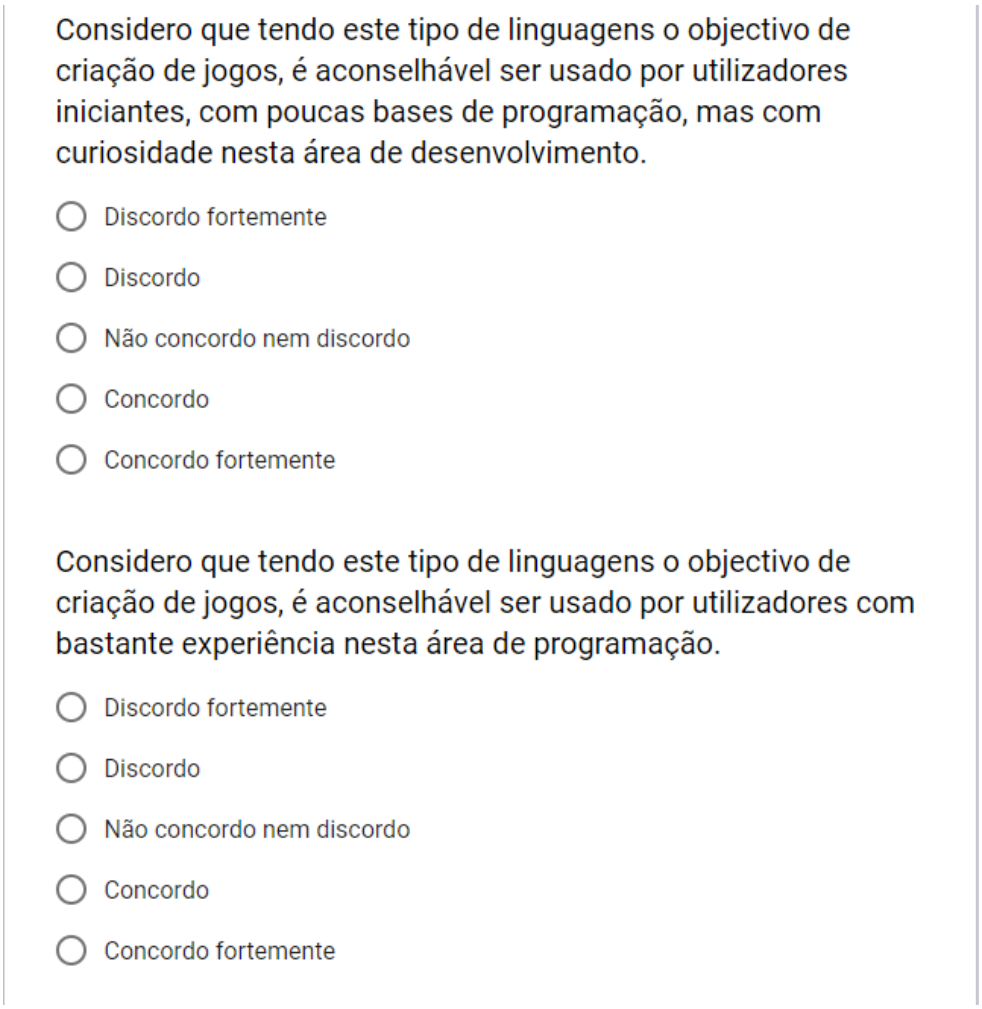


Figura A.19: Questionário Domínio de Jogos

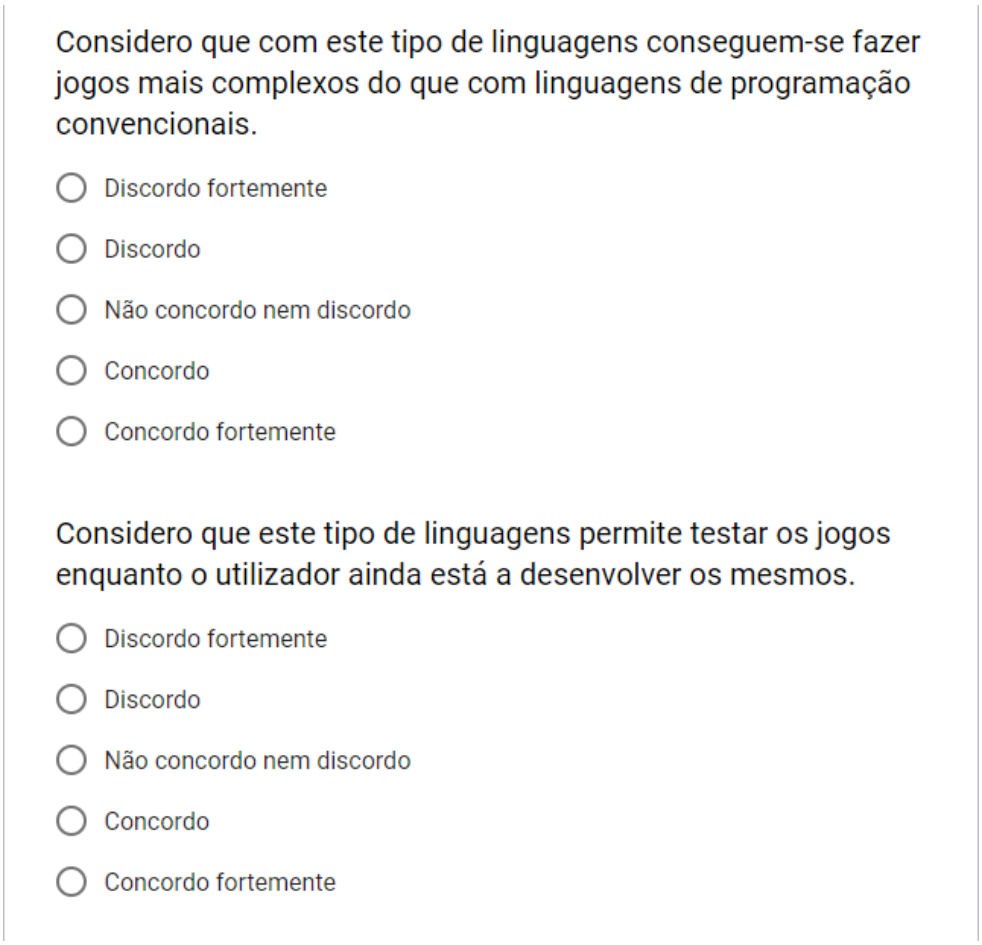


Figura A.20: Questionário Domínio de Jogos

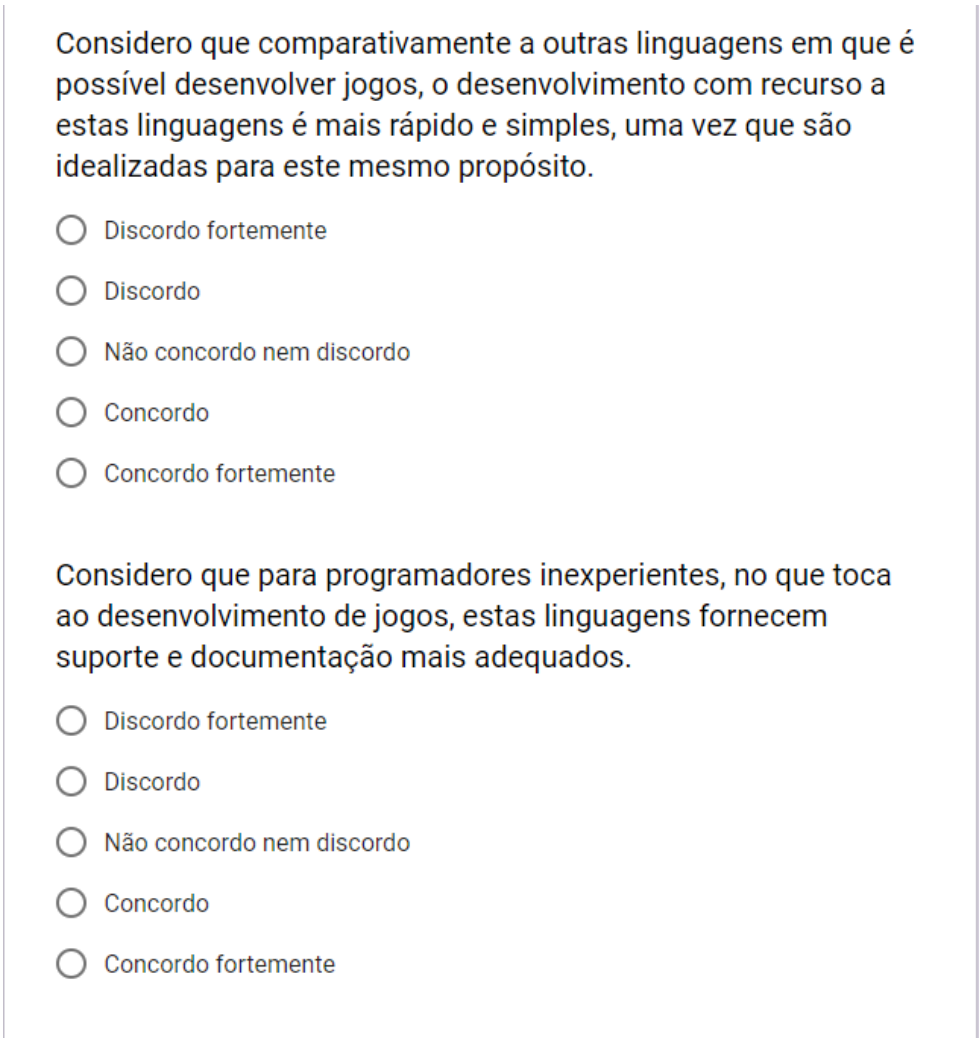


Figura A.21: Questionário Domínio de Jogos

Considero que as componentes disponibilizadas por este tipo de linguagens são autoexplicativos e não causam duvidas ao utilizador relativamente aos dados e tipos de dados que deve introduzir.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que uma das grandes vantagens deste tipo de linguagens é a fácil geração de protótipos.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo .

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Figura A.22: Questionário Domínio de Jogos

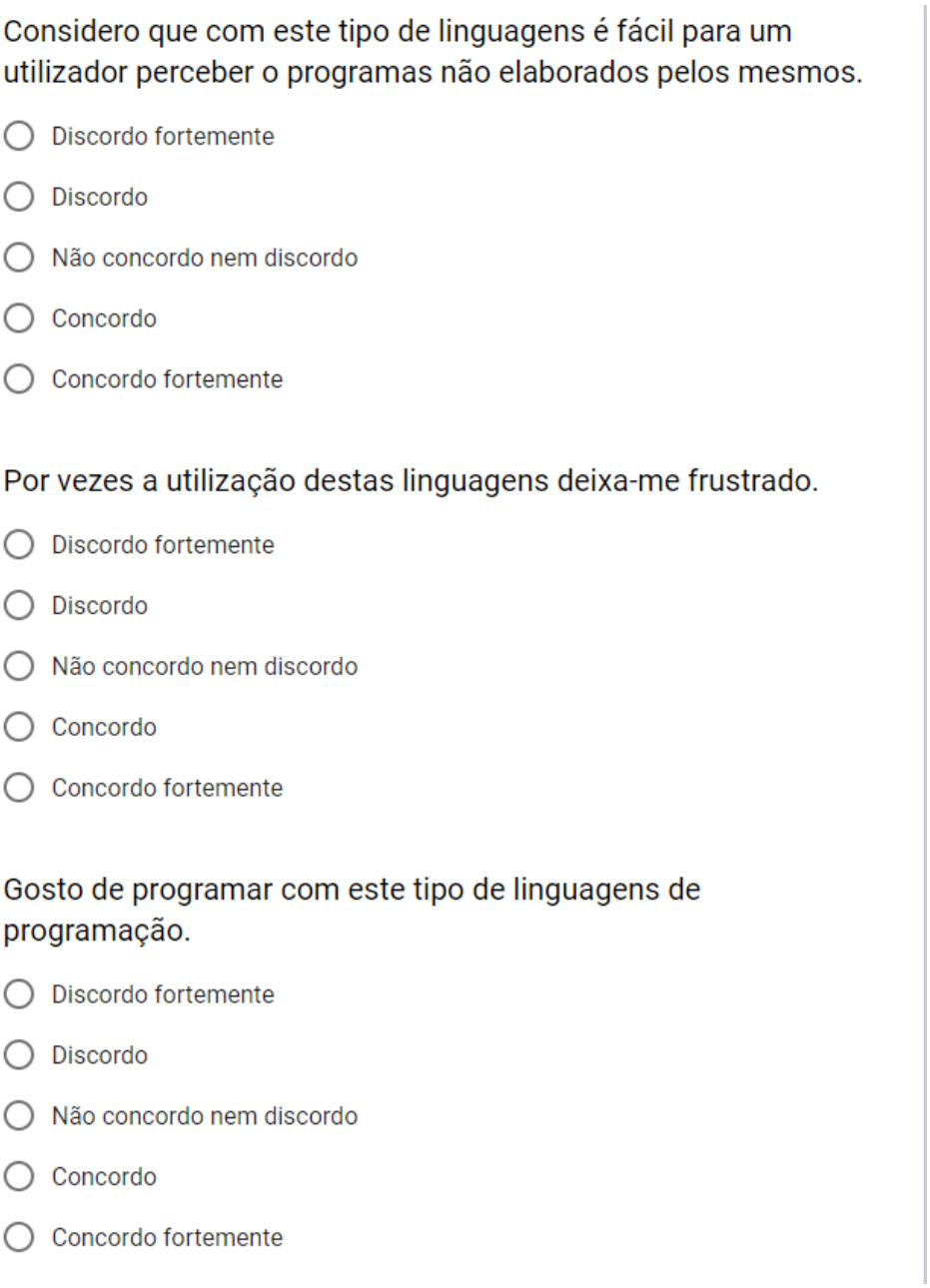


Figura A.23: Questionário Domínio de Jogos

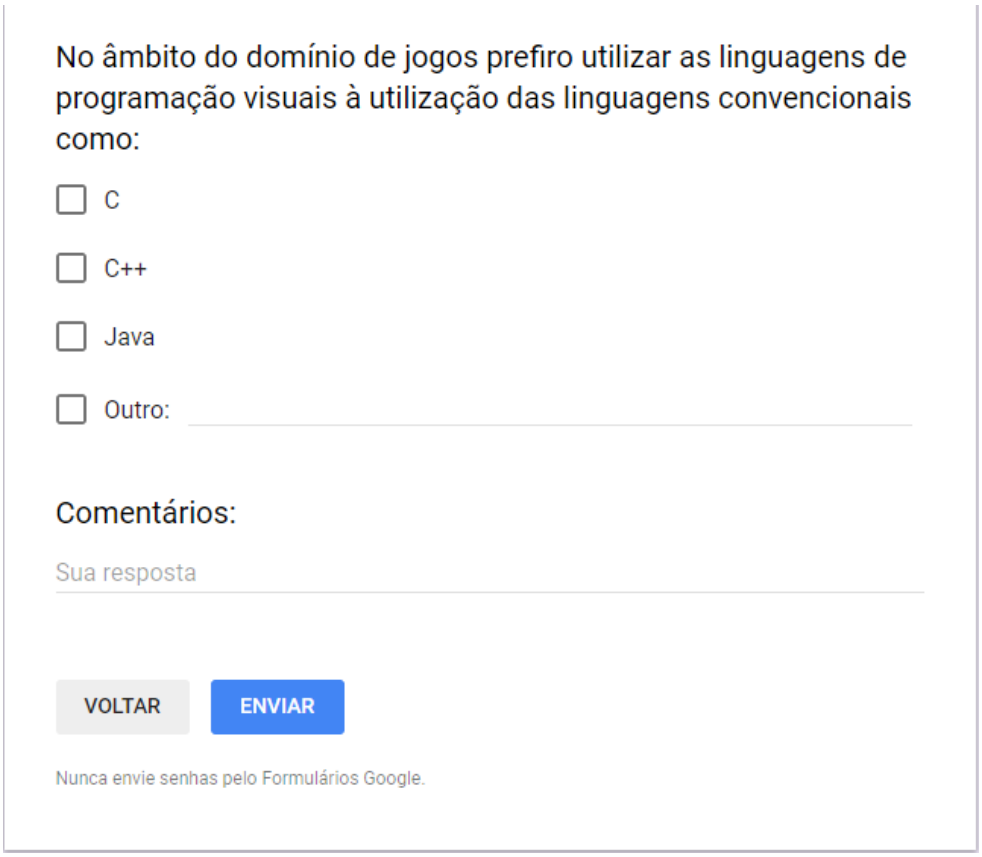
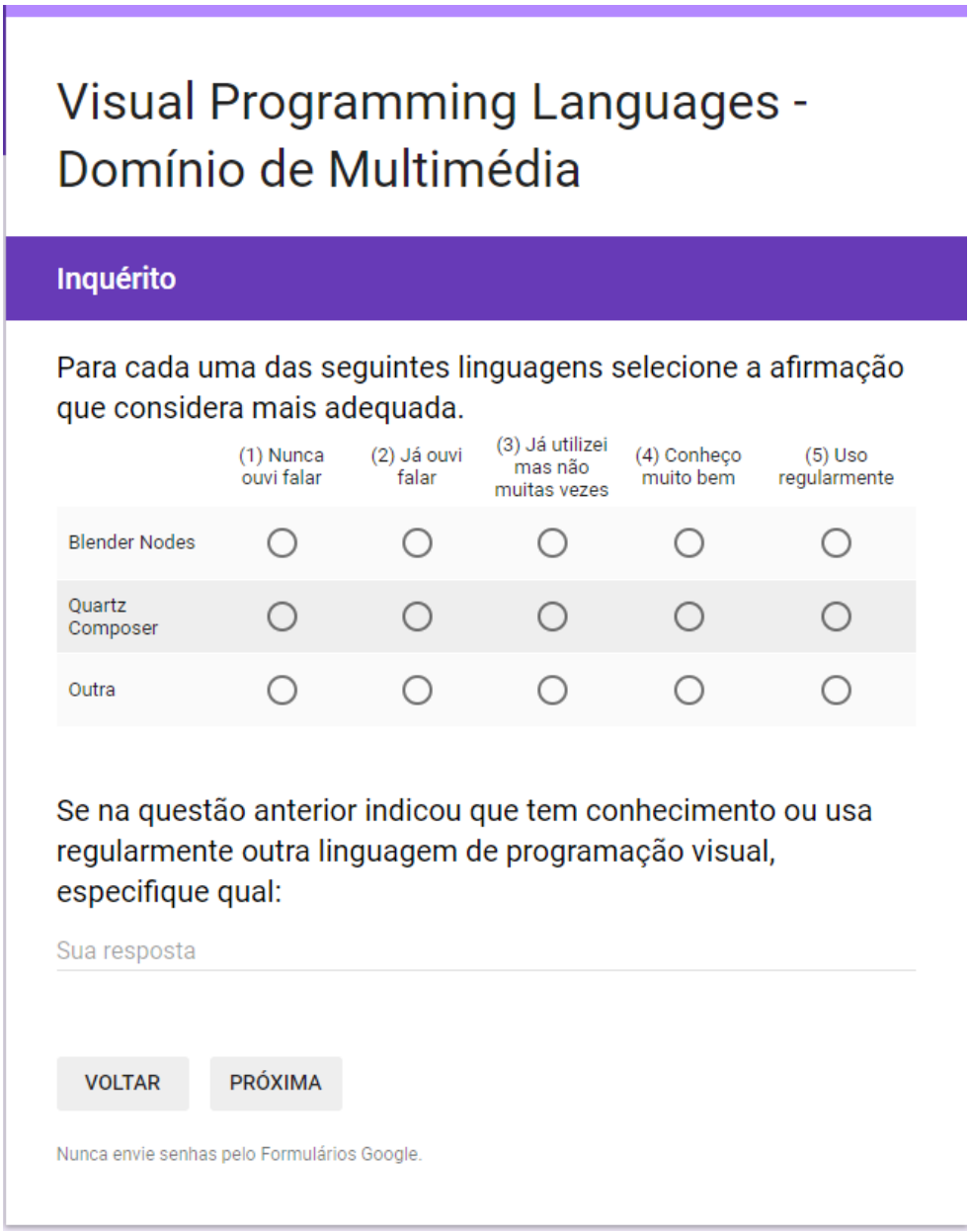


Figura A.24: Questionário Domínio de Jogos

A.3 Visual Programming Languages - Domínio Multimédia

Figura A.25: Questionário Domínio Multimédia



Visual Programming Languages - Domínio de Multimédia

Inquérito

Para cada uma das seguintes linguagens selecione a afirmação que considera mais adequada.

	(1) Nunca ouvi falar	(2) Já ouvi falar	(3) Já utilizei mas não muitas vezes	(4) Conheço muito bem	(5) Uso regularmente
Blender Nodes	○	○	○	○	○
Quartz Composer	○	○	○	○	○
Outra	○	○	○	○	○

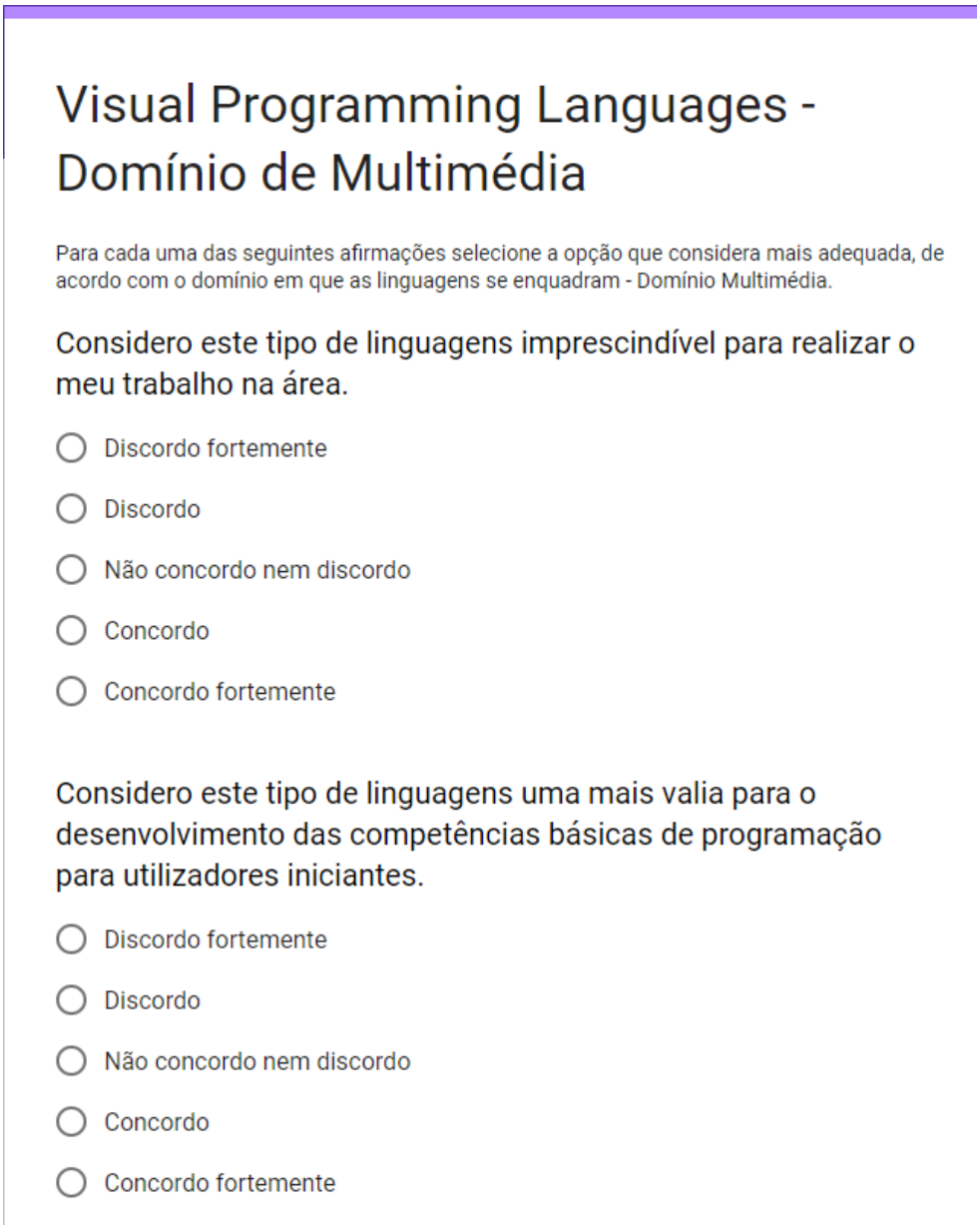
Se na questão anterior indicou que tem conhecimento ou usa regularmente outra linguagem de programação visual, especifique qual:

Sua resposta

VOLTAR PRÓXIMA

Nunca envie senhas pelo Formulários Google.

Figura A.26: Questionário Domínio Multimédia



Visual Programming Languages - Domínio de Multimédia

Para cada uma das seguintes afirmações selecione a opção que considera mais adequada, de acordo com o domínio em que as linguagens se enquadram - Domínio Multimédia.

Considero este tipo de linguagens imprescindível para realizar o meu trabalho na área.

- Discordo fortemente
- Discordo
- Não concordo nem discordo
- Concordo
- Concordo fortemente

Considero este tipo de linguagens uma mais valia para o desenvolvimento das competências básicas de programação para utilizadores iniciantes.

- Discordo fortemente
- Discordo
- Não concordo nem discordo
- Concordo
- Concordo fortemente

Figura A.27: Questionário Domínio Multimédia

Considero que este tipo de linguagens permite aos utilizadores ter uma curva de aprendizagem menor em relação às linguagens de programação convencionais.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária menor.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é mais indicado para utilizadores de uma faixa etária média-alta (18+).

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.28: Questionário Domínio Multimédia

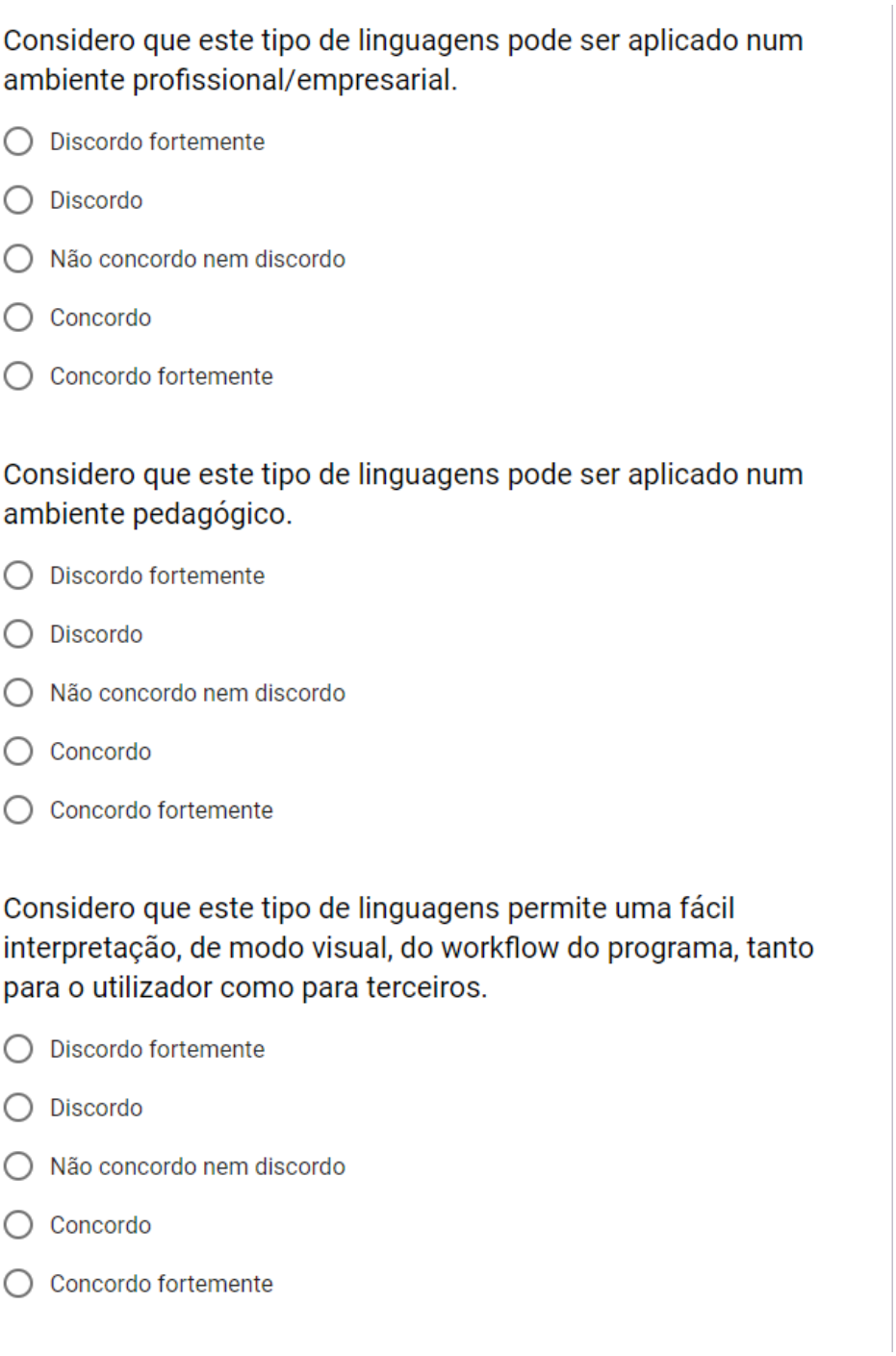


Figura A.29: Questionário Domínio Multimédia

Considero que este tipo de linguagens permite detetar facilmente erros/bugs/inconsistências no programa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens é possível ter um leque de funcionalidades equiparável a uma linguagem de programação convencional.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que este tipo de linguagens é bastante intuitivo.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.30: Questionário Domínio Multimédia

Considero que a criação de protótipos com este tipo de linguagens é mais rápida do que recorrendo às linguagens de programação convencionais.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que com este tipo de linguagens, utilizadores sem background em programação mas sim com background em arte ou música, conseguem facilmente criar os seus trabalhos.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.31: Questionário Domínio Multimédia

Considero que recorrendo a estas linguagens de programação visuais, os utilizadores criam os seus programas sem se preocuparem com a sua sintaxe, concentrando-se apenas na lógica e no fluxo de dados.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que para programadores inexperientes, no que toca ao desenvolvimento deste tipo de projetos, estas linguagens fornecem suporte e documentação bastante adequados e completos.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Figura A.32: Questionário Domínio Multimédia

Considero que, sendo este tipo de linguagens do tipo dataflow, é fácil para o utilizador perceber a lógica do seu programa bem como o seu fluxo.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que os programas elaborados com este tipo de linguagens são de fácil edição, mesmo tendo uma estrutura bastante complexa.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Considero que um utilizador que nunca tenha trabalhado com este tipo de linguagens consegue facilmente adaptar-se e perceber o que pode construir com as mesmas.

- ◯ Discordo fortemente
- ◯ Discordo
- ◯ Não concordo nem discordo
- ◯ Concordo
- ◯ Concordo fortemente

Figura A.33: Questionário Domínio Multimédia

Considero que com este tipo de linguagens é fácil para um utilizador perceber facilmente programas que já foram produzidos há algum tempo

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Considero que com este tipo de linguagens é fácil para um utilizador perceber o programas não elaborados pelos mesmos.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Por vezes a utilização destas linguagens deixa-me frustrado.

- ○ Discordo fortemente
- ○ Discordo
- ○ Não concordo nem discordo
- ○ Concordo
- ○ Concordo fortemente

Figura A.34: Questionário Domínio Multimédia

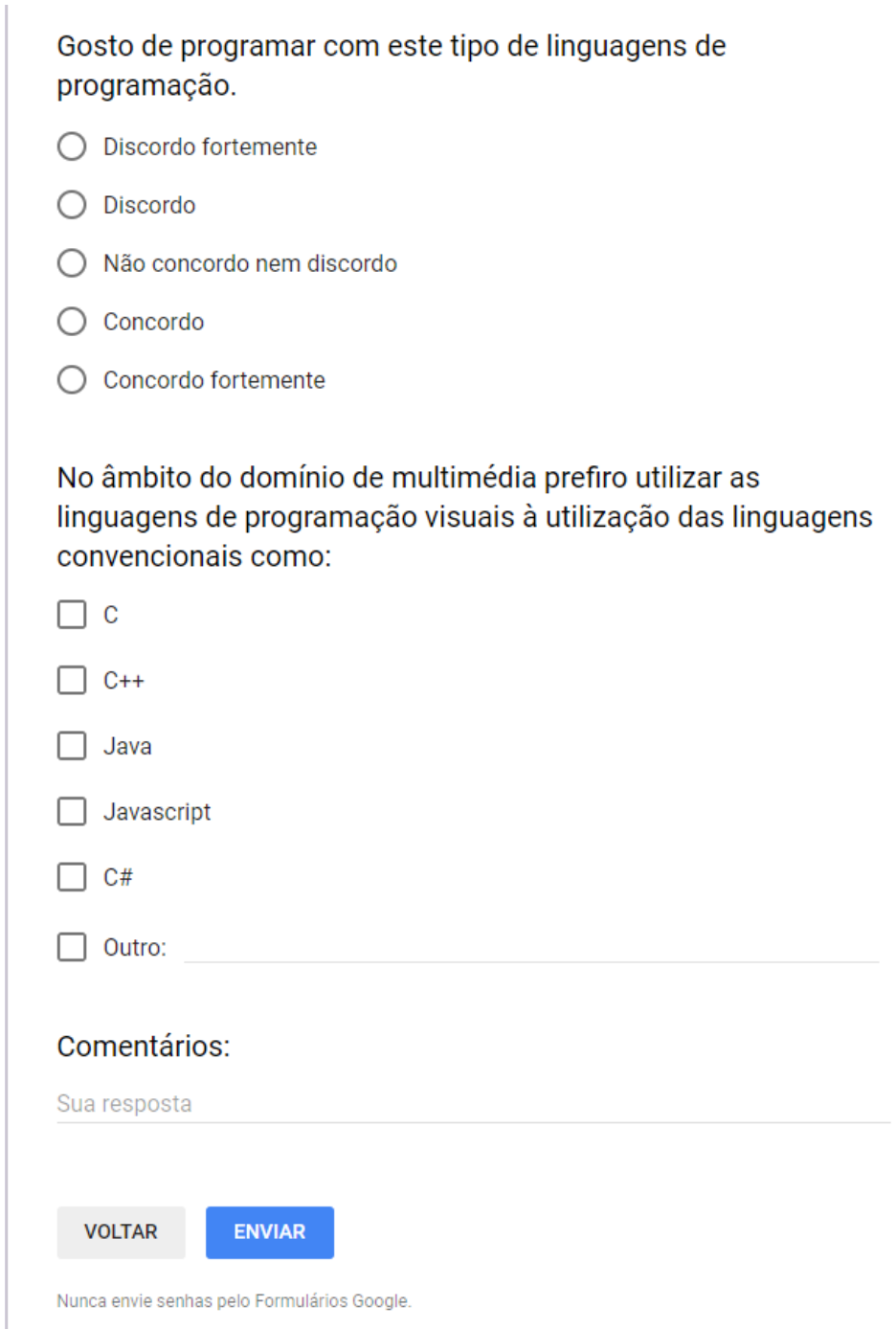


Figura A.35: Questionário Domínio Multimédia